

Japanese Patent Laid-open Publication No. H9-288564

Publication date: November 4, 1997

Applicant: SAKAMURA TAKESHI, MITSUBISHI ELECTRIC CORP

Title: DATA PROCESSING DEVICE

【Partial Translation: from Page 98, Right column, Line 16 to Line 34 】

[Mnemonic]

TRAP

[Function of instruction]

TRAP conditionally

Conditioned software interruption, TRAP

[Instruction option]

/Various condition specifications (cccc)

[Instruction bit pattern and assembler expression]

Shown in Fig. 270

[Flag change]

Shown in Fig. 271

[Explanation] When a specified condition is met, an internal interruption (trap) is performed. Since EIT is activated by the TRAP instruction, ring 0 is always entered.

The condition is specified in the same manner as that for the Bcc instruction.

According to TRAP and TRAPA, a part of PSS and only PRNG of PSM are updated, similarly to other EIT processings. Fields (including PSB) other than PRNG of PSM are not updated. When a condition undefined by TRAP is specified, it will be a reserved instruction exception (RIE).

Rest Available Conv

Fig. 270

TRAP



cccc is condition specification

Fig. 271

Instruction	F	X	V	L	M	Z	Comment
TRAP	-	-	-	-	-	-	

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-288564

(43)Date of publication of application : 04.11.1997

(51)Int.Cl.

G06F 7/38
G06F 7/02
G06F 7/50
G06F 9/305

(21)Application number : 08-241822

(71)Applicant : SAKAMURA TAKESHI
MITSUBISHI ELECTRIC CORP

(22)Date of filing : 17.06.1996

(72)Inventor : SAKAMURA TAKESHI

(54) DATA PROCESSOR

(57)Abstract:

PROBLEM TO BE SOLVED: To easily and mathematically interpret an operation result by showing the occurrence of the overflow of the operation result stored in a destination operand even if an arithmetic operation which the instruction codes of a binary number with code and the binary number without code designate is executed.

SOLUTION: A program expressed by an instruction set where the first instruction code designating an instruction for executing the arithmetic operation of addition or subtraction between two operands as the binary number with code by means of the expression of two complements and a second instruction code designating an instruction for execution the same operation as the arithmetic operation between the two operands as the binary number without code, which is absolute value-expressed, are prepared is decoded. Then, the arithmetic operations which the instruction codes designate are executed. Information showing the occurrence of overflow in the operation result stored in the destination operand by means of the respective operations is expressed by the V flag.

図表A:ADD1

	2200C	30756	88100	004/00	1.1.1	1.1.1
1	00	00	00	00	00	00
2	00	00	00	00	00	00
3	00	00	00	00	00	00
4	00	00	00	00	00	00
5	00	00	00	00	00	00
6	00	00	00	00	00	00
7	00	00	00	00	00	00
8	00	00	00	00	00	00
9	00	00	00	00	00	00
10	00	00	00	00	00	00
11	00	00	00	00	00	00
12	00	00	00	00	00	00
13	00	00	00	00	00	00
14	00	00	00	00	00	00
15	00	00	00	00	00	00
16	00	00	00	00	00	00
17	00	00	00	00	00	00
18	00	00	00	00	00	00
19	00	00	00	00	00	00
20	00	00	00	00	00	00
21	00	00	00	00	00	00
22	00	00	00	00	00	00
23	00	00	00	00	00	00
24	00	00	00	00	00	00
25	00	00	00	00	00	00
26	00	00	00	00	00	00
27	00	00	00	00	00	00
28	00	00	00	00	00	00
29	00	00	00	00	00	00
30	00	00	00	00	00	00
31	00	00	00	00	00	00

LEGAL STATUS

[Date of request for examination] 17.06.1996

[Date of sending the examiner's decision of rejection] 23.05.2000

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-288564

(43) 公開日 平成9年(1997)11月4日

(51) Int.Cl. ⁸	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F	7/38		G 0 6 F	7/38 Y
	7/02			7/02 J
	7/50			7/50 Q
	9/305			9/30 3 4 0 A

審査請求 有 発明の数 3 書面 (全 252 頁)

(21) 出願番号 特願平8-241822

(22) 出願日 平成8年(1996)6月17日

特許法第30条第1項適用申請有り 昭和62年3月31日頒布の「TRONシンポジウム資料集」に発表

(71) 出願人 593084797

坂村 健

東京都港区白金台3-12-30-105

(71) 出願人 000006013

三菱電機株式会社

東京都千代田区丸の内二丁目2番3号

(72) 発明者 坂村 健

東京都港区白金台3-12-30-105

(74) 代理人 弁理士 宮田 金雄 (外3名)

(54) 【発明の名称】 データ処理装置

(57) 【要約】

【課題】 符号付き算術演算と符号なし算術演算とを実行してもその算術結果の数学的解釈を容易に行えるようにする。

【解決手段】 2つのオペランドの間で2の補数表現による符号付き二進数として加算する演算を指定する加算命令ADD、及び2つのオペランドの間で絶対値表現された符号なし二進数として加算する演算を指定する加算命令ADDUが命令セットとして用意され、それぞれの加算演算結果のオーバーフローが発生したことを意味するフラッグをVフラッグで表現したものである。

加算命令(ADD)

本発明	VAX	IBM/370	68000	80386	32000	他
(符号付き) N(絶対値)	(符号付き) N(絶対値)	(符号付き) N(絶対値)	(符号付き) N(絶対値)	(符号付き) N(絶対値)	(符号付き) N(絶対値)	(符号付き) N(絶対値)
Z	Z	O	Z	ZF		大小
V	V	O	V	OF	F	符号
X	C	O	C	CF	C	符号なし
			X(Ck)			他
(符号なし) L(絶対値)	(符号なし) L(絶対値)	(符号なし) L(絶対値)	(符号なし) L(絶対値)	(符号なし) L(絶対値)	(符号なし) L(絶対値)	(符号なし) L(絶対値)
Z	Z	O	Z	ZF		大小
V	V					符号
X						符号なし
						他

1

【特許請求の範囲】

【請求項1】 2つのオペランド間の加算又は減算の算術演算を2の補数表現による符号付き二進数として行う命令を指定する第1の命令コードと、2つのオペランドの間の前記算術演算と同一の演算を絶対値表現された符号なし二進数として行う命令を指定する第2の命令コードとが用意された命令セットにより表現されるプログラムを解読して、前記第1および第2の命令コードの各々が指定する算術演算を実行するデータ処理装置であって、

前記第1および第2の命令コードの各々で指定される命令によりデスティネーションオペランドに格納された演算結果におけるオーバーフローの発生を示す情報をフラッグで表現すべくしてあることを特徴とするデータ処理装置。

【請求項2】 2つのオペランド間の加算又は減算の算術演算を2の補数表現による符号付き二進数として行う命令を指定する第1の命令コードと、2つのオペランドの間の前記算術演算と同一の演算を絶対値表現された符号なし二進数として行う命令を指定する第2の命令コードとが用意された命令セットにより表現されるプログラムを解読して、前記第1および第2の命令コードの各々が指定する算術演算を実行するデータ処理装置であって、

前記第1および第2の命令コードの各々で指定される命令によりなされた算術演算の演算結果の正負を示す情報をフラッグで表現すべくしてあることを特徴とするデータ処理装置。

【請求項3】 2つのオペランドの間で2の補数表現による符号付き二進数としてその大小を比較する比較演算を指定する第1の命令コードと、2つのオペランドの間で絶対値表現された符号なし二進数としてその大小を比較する比較演算を指定する第2の命令コードとが用意された命令セットにより表現されるプログラムを解読し、前記第1および第2の命令コードの各々が指定する比較演算を実行するデータ処理装置であって、前記第1および第2の命令コードの各々に従って実行された比較演算によるそれぞれの数学的大小関係の情報をフラッグで表現すべくしてあることを特徴とするデータ処理装置。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明はデータ処理装置に関し、演算結果の数学的解釈を容易にしたデータ処理装置を提案するものである。

【0002】

【従来の技術及び発明が解決しようとする課題】 従来のデータ処理装置における算術演算とフラッグ変化の関係についてみると符号付き二進数と符号なし二進数とを1種類の命令で演算し、オペランドを符号付き数と考

2

変化するフラッグと、符号なし数と考えて変化するフラッグとを各別に持っている。ところがこのような装置では、オペランドを符号付き数と考えた場合及び符号なし数と考えた場合の両方の演算結果の状態をフラッグに反映するため、オペランドを符号付き数と考えた場合に独立に意味をもつフラッグ、或はオペランドを符号なし数と考えた場合に独立に意味をもつフラッグが少なくなる。例えば、符号付き加算命令でオーバーフローが起こった場合でも結果の正負を正しく示すフラッグはなかった。また符号なし減算命令でオーバーフローが起こった場合も結果の正負を正しく示すフラッグはなかった。また独立に意味をもつフラッグが少ないため、条件によっては2つ以上のフラッグを同時に見ないと状態を判断できないという難点がある。

【0003】 次に従来装置においてデスティネーションが小さいときのフラッグ変化についてみると整数のビット列の演算結果又はソースオペランドサイズよりデスティネーションオペランドのサイズが小さいとき格納されなかったビットは無視する。又浮動小数点データの演算結果又はソースオペランドサイズよりデスティネーションオペランドのサイズが小さいとき格納されなかった下位ビットは一定規則により丸められる。このような装置には、整数演算の場合、小さなデスティネーションに格納しても値が正しく保てたかどうかをチェックするため、格納前に余分な演算を必要とした。またビット列演算の場合、格納するビット又は格納しないビットがバイト境界以外になる場合もありチェックが特に複雑となる。本発明は斯かる問題点を解決するためになされたもので、特に算術演算とフラッグ変化を密に対応づけ、演算結果の数学的解釈を容易に行えるようにしたデータ処理装置を提供することを目的とする。

【0004】

【課題を解決するための手段】 本発明は、以上の問題点を解決するために次のような手段をとっている。

1. 符号付き演算と符号なし演算とを別命令にする。
2. 全フラッグの変化を符号付き命令、符号なし命令に従い符号付き二進数として意味のあるもの、符号なし二進数として意味のあるものに専念させる。
3. 演算結果、転送結果がデスティネーションサイズに格納できないとき数学的に正しい値が保たれているかに従い、フラッグを変化させる。

【0005】 本発明に係る第1のデータ処理装置は、2つのオペランド間の加算もしくは減算の算術演算を2の補数表現による符号付き二進数として行う命令を指定する第1の命令コードと、2つのオペランド間の加算もしくは減算の算術演算を絶対値表現された符号なし二進数として行う命令を指定する第2の命令コードとのいずれが指定する算術演算を行った場合も、デスティネーションオペランドに格納された演算結果のオーバーフローの発生をフラッグで示すことを特徴とする。

【0006】本発明に係る第2のデータ処理装置は、2つのオペランド間の加算もしくは減算の算術演算を2の補数表現による符号付き二進数として行う命令を指定する第1の命令コードと、2つのオペランド間の加算もしくは減算の算術演算を絶対値表現された符号なし二進数として行う命令を指定する第2の命令コードとのいずれが指定する算術演算を行った場合も、算術演算の演算結果の正負をフラッグで示すことを特徴とする。

【0007】本発明に係る第3のデータ処理装置は、2つのオペランド間で2の補数表現による符号付き二進数としてその大小を比較する比較演算を指定する第1の命令コードと、2つのオペランド間で絶対値表現された符号なし二進数としてその大小を比較する比較演算を指定する第2の命令コードとのいずれが指定する比較演算を行った場合も、2つのオペランドの数学的大小関係をフラッグで示すことを特徴とする。

【0008】

【発明の実施の形態】以下、本発明の実施の形態につき詳しく説明する。以下の実施の形態では、

1. 四則演算、比較命令、固定長ビット操作命令をすべて符号つきと符号なしの両方用意した。
2. Lフラッグにより、符号付き加算命令と符号付き減算命令の結果の正負をオーバーフローに関係なく示す。
3. Vフラッグにより結果が格納できなかったときの数学的意味（符号付き、符号なしに応じて）を示す。

従来のプロセッサ即ち32000、80386、68000、IBM/370、VAXでは、図409、図410、図411、図412、図413、図414に示すように算術演算とフラッグ変化の関係をみると、ほとんど全てのプロセッサが符号付き数と符号なし数を1種類の命令で演算し、オペランドを符号付き数と考えて変化するフラッグと、符号なし数と考えて変化するフラッグとを共通に持っていた。

【0009】従来のプロセッサで符号なし加算命令が存在しない場合は、符号付き加算命令として演算を実行した結果のフラッグ情報を調べることによって、得られた結果が符号なし数として有効であるか否かを判断している。図415に示すように、8ビット符号なし数として192(10)、8ビット符号付き数として-64(10)を各々加算する場合に符号なし数として各々を加算した結果はオーバーフローを生じるはずであるが、符号付き数として加算された場合の結果は最上位からの桁上げビットと最上位ビットへの桁上げが一致するためオーバーフローを生じない。従って、符号なし数としてオーバーフローを調べるには、他のフラッグ、VAXの場合は桁上げフラッグがセットされていることにより検出される。但し、この場合でもオーバーフローフラッグがリセットされていることと桁上げフラッグがセットされていることの2つの条件を調べなければならず、プログラム数が増え効率が悪くなる。一方、本発明では符号付き

と符号なしの両方の命令を備えており、各々の実行結果でフラッグに反映されるので、符号なし数として扱いたい場合は符号なし加算命令を実行し、その結果オーバーフローを示すフラッグのみを参照すればオーバーフローを生じているかどうかを検出できる。また従来のプロセッサでは符号付き加算命令を実行した場合、オーバーフローが生じない時は結果の最上位ビットを見ることにより演算結果の正負が判断できたが、オーバーフローが生じた場合は結果の最上位ビットを見ただけでは演算結果の正負が判断できず、つまり結果としてのデータが無効であるという理由からオーバーフローを優先させ演算結果を無視していた。例えば図416に示すように、8ビットの負の最大数同士(-128(10))を加算した結果オーバーフローが検出されるが、結果の最上位ビットは0であり(最上位ビットからの桁上げを除く)結果が負であるということに反している。本発明では、オーバーフローが生じた時にでも演算結果の正負を情報として提供するフラッグが備わっており、そのビットを参照するだけで処理結果が明瞭である。

【0010】更に、本発明では各演算におけるオペランドサイズが異種サイズ間でも演算実行され、異種サイズ間どうしの演算においてもオーバーフロー検出される。例えば、演算結果がデスティネーションオペランドのデータサイズを越えてしまうような場合、又、ビット抽出命令によって取り出されたデータがデスティネーションオペランドのデータサイズを越えてしまうような場合にもオーバーフローを示すフラッグがセットされる。また、図417に示すように異種サイズ間どうしの符号付き加算演算においてオーバーフローが生じた場合に、デスティネーションオペランドに結果として保持されたデータ(c)の最上位ビット(d)で示される)以外に、(a)で示される演算結果の最上位ビット、つまり(b)で示されるサインビットが、正負を示すフラッグに情報として格納される(この場合は正を表す。)ので、たとえオーバーフローが生じて数値データとして意味を持たなくても演算結果のデータの正負の真偽が検出できる。

【0011】次に本発明における符号付き演算の場合の例について説明する。

ADD R0, H, R1, B

この命令は、オペランドR0をオペランドR1に符号付きで足し込む。Hは、オペランドR0がハーフワード(16ビット)であることを示す。Bは、オペランドR1がバイト(8ビット)であることを示す。

この命令の実行例を次に示す。

$$\begin{array}{r} \text{R0:} \quad 0000.0000.0111.1111 \\ \quad \quad (= +127 \quad 16 \text{ ビット符号付き}) \\ \quad \quad + \\ \text{R1:} \quad 0000.0001 \\ \quad \quad (= +1 \quad 8 \text{ ビット符号付き}) \\ \quad \quad || \\ \text{演算結果:} \quad 00 \dots\dots\dots 0.1000.0000 \\ \quad \quad (= +128 \quad 2 \text{ 進数符号付き}) \\ \quad \quad \downarrow \\ \text{演算結果:} \quad 1000.0000 \\ \quad \quad (= -128 \quad 8 \text{ ビット符号付き}) \end{array}$$

即ち、8ビット符号付きでは演算結果を表現できない。
 こういった場合にV-f l a gは1になる。

この命令の他の実行例を次に示す。

$$\begin{array}{r} \text{R0:} \quad 1111.1111.0111.1111 \\ \quad \quad (= -129 \quad 16 \text{ ビット符号付き}) \\ \quad \quad + \\ \text{R1:} \quad 0000.0001 \\ \quad \quad (= +1 \quad 8 \text{ ビット符号付き}) \\ \quad \quad || \\ \text{R0:} \quad 0000.0000.1111.1111 \\ \quad \quad (= +255 \quad 16 \text{ ビット符号なし}) \\ \quad \quad + \\ \text{R1:} \quad 0000.0001 \\ \quad \quad (= +1 \quad 8 \text{ ビット符号なし}) \\ \quad \quad || \\ \text{演算結果:} \quad 00 \dots\dots\dots 001.0000.0000 \\ \quad \quad (= +256 \quad 2 \text{ 進数符号なし}) \\ \quad \quad \downarrow \\ \text{演算結果:} \quad 0000.0000 \\ \quad \quad (= 0 \quad 8 \text{ ビット符号なし}) \end{array}$$

即ち、8ビット符号なしでは演算結果を表現できない。
 こういった場合にV-f l a gは1になる。

この命令の他の実行例を次に示す。

$$\begin{array}{r} \text{演算結果:} \quad 11 \dots\dots\dots 1.1000.0000 \\ \quad \quad (= -128 \quad 2 \text{ 進数符号付き}) \\ \quad \quad \downarrow \\ \text{演算結果:} \quad 1000.0000 \\ \quad \quad (= -128 \quad 8 \text{ ビット符号付き}) \end{array}$$

即ち、8ビット符号付きで演算結果を表現できる。こ
 ういった場合にV-f l a gは0になる。これは後のAD
 D命令の項を参照。

【0012】次に符号なし演算の場合の例について説明
 する。

ADDU R0, H, R1, R

この命令は、オペランドR0をオペランドR1に符号な
 しで足し込む。Hは、オペランドR0がハーフワード
 (16ビット)であることを示す。Bは、オペランド
 R1がバイト(8ビット)であることを示す。

この命令の実行例を次に示す。

(5)

$$\begin{array}{r} \text{R0:} \quad 0000, 0000, 0111, 1111 \\ \quad \quad (= +127 \quad 16 \text{ ビット符号なし}) \\ \quad \quad + \\ \text{R1:} \quad 0000, 0001 \\ \quad \quad (= +1 \quad 8 \text{ ビット符号なし}) \\ \quad \quad || \\ \text{演算結果: } 00 \text{ -----} 0, 1000, 0000 \\ \quad \quad (= +128 \quad 2 \text{ 進数符号なし}) \\ \quad \quad \downarrow \\ \text{演算結果:} \quad 1000, 0000 \\ \quad \quad (= +128 \quad 8 \text{ ビット符号なし}) \end{array}$$

即ち、8ビット符号なしで演算結果を表現できる。こういった場合にV-flagは0になる。これは後のADU命令の項を参照。次に符号付き転送の場合の例について説明する。

BPINSU src.H. #2. #3. base.H

この命令は、オペランドsrcの値を、オペランドbaseの中のビット番号2から4の始まる3ビット長のビットフィールドに符号付きで転送する。Hは、オペランドsrc, baseがハーフワード(16ビット)であることを示す。

この命令の実行例を次に示す。

【0013】

【数1】

$$\begin{array}{r} \text{src:} \quad 1111, 1111, 1111, 1111 \\ \quad \quad (= -1 \quad 16 \text{ ビット符号付き}) \\ \quad \quad \downarrow \\ \text{転送されるビットフィールド:} \\ \quad \quad 111 \\ \quad \quad (= -1 \quad 3 \text{ ビット符号付き}) \\ \text{base:} \quad 00 \text{ --} \text{ } 000, 0000, 0000 \\ \quad \quad (\text{下線部は転送部位を示す}) \\ \quad \quad \downarrow \\ \text{転送結果: } 0011, 1000, 0000, 0000 \end{array}$$

【0014】即ち、3ビット符号付きでは演算結果を表現できない。こういった場合にV-flagは1になる。

この命令の他の実行例を次に示す。

【0015】

【数2】

$$\begin{array}{r} \text{src:} \quad 1111, 1111, 1111, 1111 \\ \quad \quad (= -1 \quad 16 \text{ ビット符号付き}) \\ \quad \quad \downarrow \\ \text{転送されるビットフィールド:} \\ \quad \quad 111 \\ \quad \quad (= -1 \quad 3 \text{ ビット符号付き}) \\ \text{base:} \quad 00 \text{ --} \text{ } 000, 0000, 0000 \\ \quad \quad (\text{下線部は転送部位を示す}) \\ \quad \quad \downarrow \\ \text{転送結果: } 0011, 1000, 0000, 0000 \end{array}$$

【0016】即ち、この場合は3ビット符号付きで演算結果を表現できる。こういった場合にV-flagは0になる。これは後のBFINS命令の項を参照。

【0017】次に符号なし転送の場合の例について説明する。

BPINSU src.H. #2. #3. base.H

この命令は、オペランドsrcの値を、オペランドbaseの中のビット番号2から4の始まる3ビット長のビットフィールドに符号なしで転送する。Hは、オペランドsrc, baseがハーフワード(16ビット)であることを示す。

この命令の実行例を次に示す。

【0018】

【数3】

$$\begin{array}{r} \text{src:} \quad 0000, 0000, 0000, 1111 \\ \quad \quad (= +15 \quad 16 \text{ ビット符号なし}) \\ \quad \quad \downarrow \\ \text{転送されるビットフィールド:} \\ \quad \quad 111 \\ \quad \quad (= +7 \quad 3 \text{ ビット符号なし}) \\ \text{base:} \quad 00 \text{ --} \text{ } 000, 0000, 0000 \\ \quad \quad (\text{下線部は転送部位を示す}) \\ \quad \quad \downarrow \\ \text{転送結果: } 0011, 1000, 0000, 0000 \end{array}$$

【0019】即ち、3ビット符号なしでは演算結果を表現できない。こういった場合にV-flagは1になる。

この命令の他の実行例を次に示す。

【0020】

【数4】

src: 0000, 0000, 0000, 0011
(= +8 16ビット符号なし)

↓

転送されるビットフィールド:

011
(= +3 3ビット符号なし)

base: 00 __, __000, 0000, 0000
(下線部は転送部位を示す)

↓

転送結果: 0001, 1000, 0000, 0000

【0021】即ち、この場合は3ビット符号付きで演算結果を表現できる。こういった場合にV-flagは0になる。これは後のBFINSU命令の項を参照。

【0022】次に本発明におけるフラッグの構成を述べる。

P・・・Pビットエラーフラッグ

F・・・ストリング命令やキュー命令等の高機能命令の終了条件を示す

X・・・多倍長計算用の桁上がりを示す

V・・・オーバーフローが発生したことを示す

L・・・比較命令等において第1オペランドの方が小さいことを示す

M・・・演算結果のMSB (Most Significant Bit) が1であることを示す

Z・・・演算結果が0になったことを示す

従来のプロセッサで使用するCarryフラッグは、符号なし整数の大小関係を表すという意味と、多倍長演算の桁上がりを表すという意味がある。しかし、本発明ではXフラッグを備えているためCarryフラッグは整数の大小関係を表すという意味でのみ用いられる。従って、本発明ではこのCarryフラッグを大小関係を表すフラッグであると定義し、名前をLフラッグとしている。Lフラッグは符号なし演算の場合には従来のCarryフラッグと同じ振る舞いをするが、符号付き演算の場合には従来のCarryフラッグとは異なり、オーバーフローまで考慮した真の大小関係を意味する。Xフラッグは、多倍長の演算を行う場合に桁下りの状態を保持するために使用される。符号付き演算でも符号なし演算の時と同様な変化をする。これは、従来のプロセッサのCarryフラッグとほぼ同じ意味をなす。従来のプロセッサではシフト命令等ではみ出したビットを入れるためにCarryフラッグを用いていたが、本発明ではLフラッグを設けているため、はみ出したビットはXフラッグに入れることにしている。Vフラッグは、演算の結果がデスティネーションオペランドで指定されたサイズでは表現できなかったことを示す。つまり、デスティネーションオペランドのサイズの符号付き整数で表現できない時にもVフラッグがセットされる。Mフラッグと

10

Zフラッグは演算結果をデスティネーションオペランドのサイズに変換した後の値を基準にして変化する。従って、ソースオペランドのサイズよりも、デスティネーションオペランドのサイズの方が小さい場合は演算結果が0でなくてもZフラッグがセットされることがありうる。以下、符号付きと符号なしの両方を持つ演算命令各々についてのフラッグの変化について述べる。

【0023】(1) ADD命令 (符号付き) とADDU命令 (符号なし)

10 Lフラッグ: 演算結果が負になったことを示す。ADDU命令では0にリセットされる。

Mフラッグ: 演算の結果、デスティネーションオペランドに格納された最上位ビットを示す。Mフラッグも正負を表すが、オーバーフローを生じた場合正しい正負を表示しているとは言えない。

Zフラッグ: 演算の結果、デスティネーションオペランドに格納された値が0であることを示す。

Vフラッグ: デスティネーションサイズを越えるような演算結果になったことを示す。

20 Xフラッグ: デスティネーションサイズを越えて桁上げが生じたことを示す。

(2) SUB命令 (符号付き) とSUBU命令 (符号なし)

Lフラッグ: 演算結果が負になったことを示す。

Mフラッグ: 演算の結果、デスティネーションオペランドに格納された最上位ビットを示す。Mフラッグも正負を表すが、オーバーフローを生じた場合正しい正負を表示しているとは言えない。

30 Zフラッグ: 演算の結果、デスティネーションオペランドに格納された値が0であることを示す。

Vフラッグ: デスティネーションサイズを越えるような演算結果になったことを示す。SUBU命令では、結果が負になった場合も相当する。

Xフラッグ: デスティネーションサイズを越えて桁下げが生じたことを示す。

【0024】(3) MUL命令 (符号付き) とMULU命令 (符号なし)

Lフラッグ: 演算結果が負になったことを示す。MULU命令では0にリセットされる。

40 Mフラッグ: 演算の結果、デスティネーションオペランドに格納された最上位ビットを示す。Mフラッグも正負を表すが、オーバーフローを生じた場合正しい正負を表示しているとは言えない。

Zフラッグ: 演算の結果、デスティネーションオペランドに格納された値が0であることを示す。

Vフラッグ: デスティネーションサイズを越えるような演算結果になったことを示す。

Xフラッグ: 不変

(4) DIV命令 (符号付き) とDIVU命令 (符号なし)

11

Lフラッグ：演算結果が負になったことを示す。DIV命令では0にリセットされる。ただし、DIV命令において（最大負数）÷（-1）は実行した場合は0にリセットされる。又、零除算例外発生時のLフラッグは不変である。

Mフラッグ：演算の結果、デスティネーションオペランドに格納された最上位ビットを示す。Mフラッグも正負を表すが、オーバーフローを生じた場合、つまりDIV命令における（最大負数）÷（-1）を実行した際は1にセットされ、零除算例外発生時は不変である。

Zフラッグ：演算の結果、デスティネーションオペランドに格納された値が0であることを示す。ただし、DIV命令における（最大負数）÷（-1）は実行した際は0にリセットされ、零除算例外時は不変である。

Vフラッグ：デスティネーションサイズを越えるような演算結果、つまりDIV命令における（最大負数）÷（-1）の実行、及び、零除算例外を発生した場合にのみ1にセットされる。

Xフラッグ：不変

【0025】（5）MOV命令（符号付き）とMOVU命令（符号なし）

Lフラッグ：不変

Mフラッグ：演算の結果、デスティネーションオペランドに格納された最上位ビットを示す。Mフラッグも正負を表すが、オーバーフローを生じた場合正しい正負を表示しているとは言えない。

Zフラッグ：演算の結果、デスティネーションオペランドに格納された値が0であることを示す。

Vフラッグ：デスティネーションサイズを越えるような演算結果になったことを示す。（符号ビット拡張部のみ出しは許される）

Xフラッグ：不変

（6）REM命令（符号付き）とREMU命令（符号なし）

Lフラッグ：演算結果が負になったことを示す。（演算前のデスティネーションオペランドの符号をそのまま維持する）REMU命令では0にリセットされる。ただし、零除算例外発生時は不変である。

Mフラッグ：演算の結果、デスティネーションオペランドに格納された最上位ビットを示す。正負を示すフラッグであるが、零除算例外発生時は不変である。

Zフラッグ：演算の結果、0になったことを示す。ただし、零除算例外発生時は不変である。

Vフラッグ：0にリセットされる。零除算例外が生じても剰余がオーバーフローするわけではないのでクリアしておくことにすると、例外処理中でDIV命令によるエラーか、REM命令によるエラーかが判別できる。

Xフラッグ：不変

【0026】（7）CMP命令（符号付き）とCMPU命令（符号なし）

12

Lフラッグ：演算の結果、オペランド1がオペランド2よりも小さいことを示す。

Mフラッグ：不変

Zフラッグ：演算の結果、オペランド1とオペランド2が等しいことを示す。（オペランドサイズが異なる場合、サイズの大きい方に符号拡張又は、ゼロ拡張されて実行される。）

Vフラッグ：不変

Xフラッグ：不変

10 （8）BFCMP命令（符号付き）とBFCMPU命令（符号なし）

Lフラッグ：演算の結果、ビットフィールド値が小さいことを示す。

Mフラッグ：不変

Zフラッグ：演算の結果、ソース値とビットフィールドが等しいことを示す。

Vフラッグ：不変

Xフラッグ：不変

20 【0027】（9）BFEXT命令（符号付き）とBFEXTU命令（符号なし）

Lフラッグ：不変

Mフラッグ：演算の結果、デスティネーションオペランドに格納された最上位ビットを示す。正負を示すフラッグであるが、オーバーフローを生じた場合正しい正負を表示しているとは言えない。

Zフラッグ：演算の結果、デスティネーションオペランドに格納された値が0であることを示す。

Vフラッグ：デスティネーションサイズを越えるような演算結果になったことを示す。ただし、抽出されるデータが、BFEXT命令の場合に符号拡張されたデータであり拡張された部分のみがサイズをオーバーしている時はVフラッグはセットされない。同様に、BFEXTU命令の場合に抽出データがゼロ拡張されたデータであり拡張された部分のみがサイズをオーバーしている時もVフラッグはセットされない。

Xフラッグ：不変

（10）BFINS命令（符号付き）とBFINSU命令（符号なし）

Lフラッグ：不変

40 Mフラッグ：演算の結果、データ挿入フィールドに格納された最上位ビットを示す。正負を示すフラッグであるが、オーバーフローを生じた場合正しい正負を表示しているとは言えない。

Zフラッグ：演算の結果、データ挿入フィールドに格納された値が0であることを示す。

Vフラッグ：データ挿入フィールドのサイズを越えるような演算結果になったことを示す。ただし、ソースオペランドが、BFINS命令の場合に符号拡張されたデータであり拡張部分のみがサイズをオーバーしている時はVフラッグはセットされない。同様に、BFINSU命

50

令の場合にソースオペランドがゼロ拡張されたデータであり拡張された部分のみがサイズをオーバーしている時もVフラグはセットされない。

Xフラグ：不変

以下本発明装置の全貌を詳細に説明する。説明が長大になるので目次を付し、また付加的説明、説明の整理のために付録を付けた。

【0028】目次

1. 本発明装置の特徴
 - 1-1. 基本設計思想
 - 1-2. OS向きアーキテクチャ
 - 1-3. チューニングされた命令セット
 - 1-4. コンパイラ向きの命令セット
2. 本発明装置32と本発明装置64
3. 本発明装置仕様のクラス分け
4. レジスタセット
5. データタイプ
 - 5-1. ビット
 - 5-2. ビットフィールド
 - 5-3. 整数
 - 5-4. 浮動小数
 - 5-5. 10進数
 - 5-6. スtring
 - 5-7. キュー
6. 命令フォーマット
 - 6-1. 2オペランド短縮形
 - 6-1-1. レジスタ-メモリ間 (S-format, L-format)
 - 6-1-2. レジスタ-レジスタ間 (R-format)
 - 6-1-3. リテラル-メモリ間 (Q-format)
 - 6-1-4. イミディエート-メモリ間 (I-format)
 - 6-2. 1オペランド一般形 (G1-format)
 - 6-3. 2オペランド一般形
 - 6-3-1. 第一オペランドはメモリ読みだし (G-format)
 - 6-3-2. 第一オペランドは8ビットイミディエート (E-format)
 - 6-3-3. 第一オペランドはアドレス計算 (GA-format)
 - 6-3-4. その他の2オペランド命令
 - 6-4. ショートブランチ
 - 6-5. その他
7. アドレスリングモード
 - 7-1. Pビット
 - 7-2. フォーマット中で使われる記号
 - 7-3. レジスタ直接
 - 7-4. レジスタ間接
 - 7-5. レジスタ相対間接

- 7-6. イミディエート
- 7-7. アブソリュート
- 7-8. PC相対間接
- 7-9. スタックポップ
- 7-10. スタックプッシュ
- 7-11. レジスタ相対付加モード
- 7-12. PC相対付加モード
- 7-13. 絶対付加モード
- 7-14. FP相対間接
- 7-15. SP相対間接
- 7-16. 付加モードのフォーマット
- 7-17. 付加モード仕様のレベル
8. インプリメント関連事項
 - 8-1. 仮想記憶のサポート
 - 8-2. プログラムによる命令の書き換え
9. EIT処理
10. PSWの構成
 - 10-1. PSSの構成
 - 10-2. PSHの構成
 - 10-3. フラッグの変化
11. 命令セットの記述について
 - 11-1. 記述形式の概要
 - 11-2. 命令ビットパターンとアセンブラ表記
 - 11-3. フィールド名
 - 11-4. オペランドフィールド名
 - 11-5. アドレスリングモードに関する制限
 - 11-6. 解説に関する注意
12. 本発明装置の命令セット
 - 12-1. データ転送命令
 - 12-2. 比較、テスト命令
 - 12-3. 算術演算命令
 - 12-4. 論理演算命令
 - 12-5. シフト命令
 - 12-6. ビット操作命令
 - 12-7. 固定長ビットフィールド操作命令
 - 12-8. 任意長ビットフィールド操作命令
 - 12-9. 10進演算命令
 - 12-10. String命令
 - 12-11. キュー命令
 - 12-12. ジャンプ命令
 - 12-13. マルチプロセッサ用の命令
 - 12-14. 制御空間、物理空間操作命令
 - 12-15. OS関連命令
 - 12-16. MMU関連命令
- 付録1. 本発明装置命令セットレファレンス
- 付録2. 本発明装置のアセンブラ表記について
- 付録3. 本発明装置メモリ管理方式概要
- 付録4. 本発明装置のフラッグ変化
- 付録5. 異種サイズ間の演算について
- 付録6. 高級言語向きサブルーチンコール

付録7. 制御レジスタと制御空間

付録8. 本発明装置のCTXB

付録9. 本発明装置のEIT処理

付録10. 本発明装置の命令セットパターン

付録11. 高機能命令の詳細仕様と終了時のレジスタ値

付録12. オペランドが干渉した場合の動作

付録13. キャッシュやTLBの整合性確保について

【0029】1. 本発明装置の特徴

【0030】1-1. 基本設計思想

・本発明装置はRISCではない。基本命令の高速実行を第一目標とし、さらに高機能命令を追加した。

・32ビット版の本発明装置32と64ビット版の本発明装置64を同時に設計し、32ビット版と64ビット版のチップをシリーズ化した。したがって、64ビットデータ、64ビットアドレッシングへの拡張が始めから考慮されている。

・OSと込みで開発し、リアルタイムOSであるITRON (Industrial-TRON) とワークステーション用のOSであるBTRON (Business-TRON) を高速で実行することを目指した。本発明装置はTRON (LIR) 仕様を満たし、特に実記憶環境での高速処理に重点を置く。

・将来のASIC LSIの核となるマイクロプロセッサとする。

【0031】1-2. OS向きアーキテクチャ

・ビットマップ操作サポート命令

BTRONで必要となるビットマップの移動、演算を行なう命令

・コンテキストスイッチ命令

ITRONにおいて高速のタスク切り替えを行なうための命令

・キュー操作命令

ITRONのレディキュー、ウェイトキューの操作を行なう命令

・2レベルのリング保護によるメモリ管理 (将来の拡張用にさらに2レベルのリングを用意している。)

【0032】1-3. チューニングされた命令セット

・頻度の高い命令、アドレッシングモードが短い命令となるようにチューニングレジスタ間演算、リテラル演算の命令長を短縮

【0033】1-4. コンパイラ向きの命令セット

・直交化された命令セット

・データの保持、アドレスの保持、インデクス値の保持といった各種の目的に使用できる16本の汎用レジスタ

・強力なアドレッシングモード

付加モードにより、任意段数のインデクス加算と間接参照が可能。

・異なるデータサイズ間での演算が可能

ソースオペランドとデスティネーションオペランドのサイズを別々に指定可能。

・高級言語向きの高機能ジャンプ命令

【0034】2. 本発明装置32と本発明装置64

本発明装置では、32ビット版の本発明装置32と64ビット版の本発明装置64をシリーズ化して扱っており、64ビット版への拡張が始めから考慮されているのが大きな特徴である。本発明装置64では、64ビットのリニアアドレスの空間が提供される。また、本発明装置64では、本発明装置32で扱うデータタイプに加えて、64ビット整数を扱うことができる。本発明装置64での32ビット/64ビット切り替え方法は次のようになっている。

・オペランドのデータサイズについて

各命令、各オペランド毎に存在するサイズ指定ビットにより、命令単位、オペランド単位で32ビット/64ビットを選択する。データサイズの場合は、32ビット、64ビットのほかに8ビット、16ビットも利用できるので、4つのサイズの選択を2ビットのフィールドで指定する。本発明装置32では、64ビットデータを扱わず、64ビットのデータサイズを指定した命令はエラーとする。

・ポインタのサイズについて

通常は本発明装置32で32ビットポインタ、本発明装置64で64ビットポインタを使用するが、本発明装置64で本発明装置32用のオブジェクトコードを実行するため、本発明装置64にはポインタサイズを32ビットにするモードを設ける。このモードはPSW中で指定されるので、コンテキスト (プロセスやタスク) 単位で32ビット用のプログラムと64ビット用のプログラムを混在させることは可能である。このほか、64ビットアドレッシングを行なうための拡張用ビットとして、メモリアクセスを伴うオペランド毎に「Pビット」と呼ばれる予約ビットが設けられている。ポインタサイズの32ビット/64ビット切り替えを命令単位とせず、モードにしたのは、次のような理由による。ポインタの場合、異種のサイズのものを混在させることは本質的に無理がある。というのは、ポインタというのは場所を区別するものであるため、一つでも64ビットのものがあれば、全体を64ビットのポインタにしないと区別できないからである。したがって、32ビットポインタと64ビットポインタを命令毎に切り替え、混在できるようにしたとしても、大部分はコンテキスト単位で同じ指定を繰り返すだけになり、ビット割り当てとしては効率の悪いものになる。そのような状況では、モードの方が適当である。

【0035】モードを使って32/64を切り替える場合、モードをいつ設定するか、本発明装置32と本発明装置64の互換性が大丈夫かといった疑問が生ずるかもしれない。しかし、デフォルトが32ビットポインタとなるようにしておき、64ビットアドレスを使用する時にモード変更するという形態にすれば、本発明装置64

でも本発明装置32用のプログラムをそのまま走らせることができる。また、32ビットポインタと64ビットポインタの切り替えをモードではなく命令単位にしたとしても、OSがスタックをどこに設定するか、システムコールのパラメータが32ビット/64ビットのどちらか、といったことを判断するために、OSは各コンテキストが32ビットか64ビットかということを認識しておく必要がある。その場合には、(スタックに退避された)PSW中のモードを見て32ビット/64ビットを判断できる方が便利なおことがある。

【0036】3. 本発明装置仕様のクラス分け
本発明装置では、64ビット版への拡張、シリーズ化、多様な用途への適応、などといった要求に対応するため、インプリメントするかどうかオプションとなっている機能がある。このような「オプション機能」の位置付けを明確にするため、本発明装置の仕様を次のようにクラス分けする。

〈〈L0〉〉仕様 (Level 0)

本発明装置として必ず満たさなければならない仕様である。〈〈L0〉〉仕様の例は、ユーザプログラムから見たプログラミングモデル (ISPの大部分、汎用レジスタ、PSH)、機械語のビットパターンなどである。仕様書中では、特に何も書いてない部分が〈〈L0〉〉仕様となる。

〈〈L1〉〉仕様 (Level 1)

インプリメントしておくのが原則であるが、特に用途の限定された軽い仕様のプロセッサを作りたい場合には、必ずしもインプリメントしなくてもよい仕様である。

〈〈L1〉〉仕様になるのは、ストリング命令、付加モード、キュー操作命令、ビットマップ命令といった高機能命令などであるが具体的にどの命令を〈〈L1〉〉仕様とするかは別に定める。

【0037】〈〈L1R〉〉仕様 (Level 1 Real)

〈〈L1〉〉仕様から命令再実行関係の機能とMMU関係の機能を削除した仕様であり、ITRONとμBTRONなどを実記憶ベースで効率よく動かすための仕様である。〈〈L1R〉〉の命令セットは〈〈L1〉〉とほぼ同じであり、コンパイラやユーザプログラムは共通に使用できるようにする。ただし、MMU関係 (MOVP Aなど)、OS関係 (JRNGなど) の一部の命令については、サポートしないものがある。

〈〈L2〉〉仕様 (Level 2)

将来のハードウェア量の増加にしたがって導入される予定の仕様である。命令の対称性を高めるための仕様と、演算の高速化に対応して新たに追加する命令の仕様とがある。前者の例としてはBVSC命令の' / B' オプション、ストリング命令での複雑な終了条件、無限段数の付加モードなどがあり、後者の例としてはINDEX命令などがある。仕様書中では、〈〈L2〉〉仕様の部

分を〈〈L2〉〉で示す。

【0038】〈〈LX〉〉仕様 (eXtension)
本発明装置64への拡張にしたがって導入される予定の仕様である。L2仕様と同様の意味を持つが、本発明装置64への対応ということで別のクラスとして扱う。

〈〈LX〉〉仕様の例は、64ビット演算命令などである。仕様書中では、〈〈LX〉〉仕様の部分を〈〈LX〉〉で示す。

〈〈LU〉〉仕様 (Undefined)

10 将来の拡張によって導入される予定の仕様であるが、現段階ではまだ具体的な仕様まで提示されていないものである。

〈〈LV〉〉仕様 (Variable)

各メーカーが全く自由に仕様を決めてよい部分である。チップのピン配置、パイプラインの段階や性能に関する仕様、各メーカーに割り当てられた命令のビットパターン、制御レジスタの使い方などが〈〈LV〉〉仕様の例である。このうち、各メーカーに割り当てられた命令ビットパターンについては、ビットパターンのレファレンス中でLVreservedにより示されている。

【0039】〈〈LA〉〉仕様 (Alternative)

本発明装置としての標準仕様が提示されている (あるいは、提示される予定がある) が、他にやむを得ない理由があれば変更してもよいという仕様である。もちろん、仕様を変更した部分に関しては互換性の失われる場合がある。〈〈LA〉〉仕様は、TRONとしての互換性を保証しない仕様である。〈〈LA〉〉仕様の例は、メモリ管理方式、制御レジスタと特権命令の一部などであり、主にOSの関係する部分である。本発明装置はMMUを内蔵せず、特に実記憶環境での高速処理を目指す。従ってメモリ管理に関する〈〈LA〉〉仕様の大部分は本発明装置でサポートされない。

【0040】4. レジスタセット

・本発明装置32では32ビット長の汎用レジスタが16本、本発明装置64では64ビット長の汎用レジスタが16本存在する。

・スタックポインタ (Stack Pointer-S P)、フレームポインタ (Frame Pointer-F P) は汎用レジスタに含まれる。SPはR15、FPはR14となる。

・プログラムカウンタ (Program Counter-PC) は汎用レジスタに含まれない。

・汎用レジスタは、データ保持、ベースアドレス保持、あるいはインデックスレジスタとして、各種の目的に使用できる。

・プロセッサの状態を保持するレジスタ (Processor Status Word-PSW) を持つ。図1、図2は、本発明装置64の場合〈〈LX〉〉のレジスタセットを示す。

19

・SPはコンテキスト（リング番号、割り込み処理中）に応じて切り替わる。

・PSWは4バイトからなる。下位第一バイトがステータスの表示（Processor Status Byte-PSB）、下位第二バイトがユーザのモード設定（PSBと合わせてProcessor Status Halfword-PSH）、上位の2バイトがシステムの状態表示用、となる。

・本発明装置はいわゆるbig-endianのチップであり、レジスタ上のデータについては、8ビット、16ビットのデータをLSB側に詰めて配置する。したがって、データサイズとは無関係な絶対的なビット番号を定義することができない。ビット番号を議論する場合には、必ずデータサイズと組にして扱う必要がある。これを「ビット位置」と呼ぶ。

・レジスタ上の8ビットデータに対しては、ビット位置はMSB側から0, 1, . . . 7と付けられる。また、レジスタ上の16ビットデータに対しては、ビット位置はMSB側から0, 1, . . . 15と付けられ、レジスタ上の32ビットデータに対しては、ビット位置はMSB側から0, 1, . . . 31と付けられる。したがって、8ビットデータのビット位置7のビット、16ビットデータのビット位置15のビット、32ビットデータのビット位置31のビットが物理的には同一のビットとなる。

・レジスタをデスティネーションオペランドとする命令において、レジスタ側のデータサイズが8ビット、16ビットであった場合には、上位バイトは影響を受けず、無変化となる。これは、メモリ上で演算を行なった場合の仕様と合わせたものである。上位ビットにまで影響を与えたい場合は、異種サイズ間の演算を利用する。

【例】

```
MOV    #H'12345678, R0.W
MOV    #H'aa, R0.B
R0=H'123456aaとなる。
```

・レジスタ上に8ビット、16ビットのデータを置く場合には、LSB側に詰められるので、例えば、

```
MOV.W  #H'12345678, R0
MOV.B  #H'aa, R0
MOV.W  R0, R1
```

の結果はR1=H'123456aaとなる。一方、メモリに対して同じことを行なった場合

```
MOV.W  #H'12345678, @R0
MOV.B  #H'aa, @R0
MOV.W  @R0, R1
```

には、8ビット、16ビットのデータのMSB側が揃うことになるので、R1=H'aa345678となる。

20

レジスタ上とメモリ上で結果が異なるので、注意が必要である。

【0041】5. データタイプ

本発明装置では、いわゆるbig-endianを採用している。すなわち、バイトアドレスの指定、ビット番号の指定とも、小さい番号（アドレス）の方がMSB

（Most Significant Bit/Byte）となっている。big-endianでは、メモリ上のあるデータについて、それを8ビットデータとして見る時と16（32）ビットデータとして見る時のアドレスが異なってくるため、注意が必要である。例えば、

```
address:  N    N+1    N+2    N+3
data:      0      0      0      H'12
```

といった場合に、32ビットデータとしてのアドレスNの内容はH'00000012であるが、（H'は16進を表わす）、同じ内容のデータを8ビットデータとして扱うときは、アドレスN+3を参照しなければならない。

【0042】ただし、レジスタ上のデータに関しては、8ビットデータ、16ビットデータがLSB側に詰めて配置されるため、レジスタ上に置かれたデータを、そのまま別のサイズのデータとして扱うことができる。例えば、

```
MOV    #0, R0.W
MOV    #H'12, R0.B
MOV    R0.W, R1.W
```

の結果はR1=H'00000012となる。（命令の意味については本文参照）

一方、メモリに対して同じことを行なった場合

```
MOV    #0, @R0.W
MOV    #H'12, @R0.B
MOV    @R0.W, R1.W
```

には、8ビットデータH'12と32ビットデータのMSB側が揃うことになるので、R1=H'12000000となる。本発明装置でサポートしているデータタイプを以下に説明する。

【0043】5-1. ビット

図3のように太線内が対象ビットである。メモリ上のビット操作の場合、offsetは任意レジスタ上のビット操作の場合、offsetは一つのレジスタ内に限定（offsetの上位ビットを無視する）

ビットの指定は、base_addressの指定、base_addressのサイズの指定、offsetの指定の組によって行なわれる。メモリ上のビットを対象とした場合には、base_addressで示されるメモリアドレスのMSBがoffset=0のビットとなる。この時、base_addressのサイズの

指定は、実際に操作されるビットには影響しない。ビット操作命令では、メモリに対してread-modify-writeを行なうアクセスサイズを指定するためにbase_addressのサイズの指定が利用されるが、アクセスサイズが異なっても実際に操作されるビットは同じである。一方、レジスタ上のビットを対象とした場合には、base_addressのサイズとして指定されたデータサイズでのMSBがoffset=0のビットとなる。base_addressのサイズが異なれば、実際に操作されるビットも異なったものになるので、注意が必要である。

【0044】5-2. ビットフィールド

・符号付きビットフィールド

図4に示すように太線内が対象ビットフィールドである。

$0 < width \leq 32$ (<<LX>> $0 < width \leq 64$)

S:符号ビット

base_addressのMSBから、対象ビットフィールドのMSB (符号ビット) までのビットの隔たりがoffsetとなる。

BF: G命令によるメモリ上のビットフィールド操作の場合、offsetは任意。

BF: E命令によるメモリ上のビットフィールド操作、およびレジスタ上のビットフィールド操作の場合、base_addressの1ワード (1ロングワード) をはみ出した部分のビットフィールドについて、動作を保証しない。

・符号なしビットフィールド

図5に示すように太線内が対象ビットフィールドである。

$0 < width \leq 32$ (<<LX>> $0 < width \leq 64$)

base_addressのMSBから、対象ビットフィールドのMSBまでのビットの隔たりがoffsetとなる。

BF: G命令によるメモリ上のビットフィールド操作の場合、offsetは任意。

BF: E命令によるメモリ上のビットフィールド操作、およびレジスタ上のビットフィールド操作の場合、base_addressの1ワード (1ロングワード) をはみ出した部分のビットフィールドについて、動作を保*

ー オペコードの入る部分

リテラル、またはイミディエート値の入る部分

Ea 8ビットで指定する一般形のアドレッシングモード

Sh 6ビットで指定する短縮形のアドレッシングモード

Rn レジスタの指定を行なう部分

フォーマットの記述は、右側がLSB側で、かつ高いアドレスになっている。(big-endian) フォーマットの記述例を図13に示す。アドレスNとア

*証しない。

・任意長ビットフィールド

offset、widthとも任意。ただしwidth > 0。

【0045】5-3. 整数

図6、図7に整数のデータタイプを示す。

【0046】5-4. 浮動小数

浮動小数点の演算は、コプロセッサで扱う。浮動小数点の形式はIEEE規格である。詳細は別に定める。

・単精度32ビット浮動小数 (<コプロセッサ>)

・倍精度64ビット浮動小数 (<コプロセッサ>)

・80ビット浮動小数 (<コプロセッサ>)

【0047】5-5. 10進数

多倍長の10進数の四則演算は、コプロセッサで扱う。本発明装置のメインプロセッサでは、以下に示すような固定長の符号なしPACKED形式10進数、および符号付きPACKED形式10進数を扱う。ただし、符号付きPACKED形式10進数を扱う命令は、すべて<<L2>>である。図8、図9にデータタイプをしめす。

【0048】5-6. スtring

図10、図11にstringの場合のデータタイプを示す。

【0049】5-7. キュー

図12にダブルリンクでつながれた線形リストのデータタイプを示す。

【0050】6. 命令フォーマット

命令は16ビット単位で可変長となっており、奇数バイト長の命令はない。2オペランド命令には、大きくわけて、4バイト+拡張部の構成をもち、すべてのアドレッシングモード (Ea) が利用できる一般形、および頻度の高い命令とアドレッシングモード (Sh) のみを使用できる短縮形、の2つのフォーマットがある。必要となる命令機能とコードサイズに合わせて、より適した方を選択することができる。

【0051】本発明装置の命令フォーマットは、細かい点まで気をつければかなり多くの種類に分かれる。しかし、理解を容易にするため、ここでは本発明装置の命令フォーマットを大まかに分類して説明を行なう。命令フォーマットの詳細については、付録10を参照のこと。フォーマット中に現われる記号の意味は次の通りである。

ドレスN+1の2バイトを見ないと命令フォーマットが判別できないようになっているが、これは、命令が必ず16ビット (2バイト) 単位でフェッチ、デコードされ

23

ることを前提としたためである。いずれのフォーマットの場合も、各オペランドのE aまたはS hの拡張部は、必ずそのE aまたはS hの基本部を含むハーフワードの直後に置かれる。これは、命令により暗黙に指定されるイミディエートデータや、命令の拡張部に優先する。したがって、4バイト以上の命令では、E aの拡張部によって命令のオペコードが分断される場合がある。また、付加モードなどによって、E aの拡張部にさらに拡張部 * 命令の第一ハーフワード

(Ea1 の基本部を含む)

Ea1 の拡張部

Ea1 の付加モード拡張部

命令の第二ハーフワード

(Ea2 の基本部を含む)

Ea1 の拡張部

命令の第三ハーフワード

の順となる。

【0052】なお、アラインメントの関係で16ビットのフィールドのうちの8ビットのみを使用するケースでは、使用する8ビットは下位順（アドレスの大きい方）に詰めて置かれるものとする。これに該当するのは、オペランドサイズが8ビットで、E a R, S h Rに# i m m _ d a t aのモードを指定した場合、I - f o r m a tでオペランドサイズが8ビットの場合、B R A : G, B c c : G, B S R : GでS S = 0 0の場合、などである。例えば、

MOV: I. B #H' 12, @R0

の場合、第一バイトがMOV: I. Bのオペコード、第二バイトがオペコードの一部とS h W (@R0)の指定、第三バイトは0、第四バイトがH' 12となり、ビットパターンは図14のようになる。この場合、16ビットのフィールドの上位側（アドレスの小さい方）の8ビットには必ず0を入れておかなければならない。上位8ビットが0でない場合は、これによって表現されるデータがインプリメント依存の不定値になるものとする。つまり、I - f o r m a t, # i m m _ d a t aモードの場合はそのオペランドがインプリメント依存の値になり、B R A : G, B c c : G, B S R : G命令の場合は、ジャンプ先が不定となる。いずれの場合も、E I T (例外) とはしない。

【0053】6-1. 2オペランド短縮形

【0054】6-1-1. レジスターメモリ間 (S - f o r m a t, L - f o r m a t)

図15にその例を示す。L - f o r m a t, S - f o r m a tの命令には、サイズ指定のできるもの (MOV: L, MOV: S, CMP: L) とサイズ指定のできないもの (ADD: L, SUB: L) がある。サイズ指定のできる命令では、R R等によるサイズ指定はメモリ側の

24

*が付く場合にも、次の命令オペコードよりもそちらの方が優先される。例えば、第一ハーフワードにE a 1を含み、第二ハーフワードにE a 2を含み、第三ハーフワードまである6バイト命令の場合を考える。E a 1に付加モードを使用したため、普通の拡張部のほかに付加モード拡張部もつくものとする。この時、実際の命令ビットパターンは

みに適用され、レジスタ側のサイズは32ビット固定となっている。レジスタ側とメモリ側のサイズが異なる場合には、ソース側のサイズが小さい場合に符号拡張が、デスティネーション側のサイズが小さい場合に上位バイトのカットとオーバーフローのチェックが行なわれる。一方、サイズ指定のできないADD: L, SUB: L命令では、レジスタ側、メモリ側のオペランドサイズとも32ビット固定である。レジスタ側のサイズを32ビット固定としたのは、本発明装置において、「レジスタ上のデータは、できる限り32ビット符号付き整数として扱う」という原則を設けているためである。この原則は、L - f o r m a t, S - f o r m a t命令のほか、ビットフィールド命令や高機能命令でレジスタ上にオペランドを置く場合にも適用される。

【0055】6-1-2. レジスターレジスタ間 (R - f o r m a t)

図16にその例を示す。

【0056】6-1-3. リテラルメモリ間 (Q - f o r m a t)

図17にその例を示す。

【0057】6-1-4. イミディエートメモリ間 (I - f o r m a t)

図18にその例を示す。I - f o r m a tのイミディエート値のサイズは、デスティネーション側のオペランドのサイズと共通に8, 16, 32, 64ビットとなり、ゼロ拡張、符号拡張は行なわれない。

【0058】6-2. 1オペランド一般形 (G 1 - f o r m a t)

図19にその例を示す。

【0059】6-3. 2オペランド一般形

ここに含まれるのは、8ビットで指定する一般形アドレスレジスタモードのオペランドが2つ存在する命令であ

25

る。オペランドの総数は3つ以上になる場合がある。

【0060】6-3-1. 第一オペランドはメモリ読みだし (G-format)

図20にその例を示す。

【0061】6-3-2. 第一オペランドは8ビットイミディエート (E-format)

図21にその例を示す。このフォーマットとイミディエートメモリ間のフォーマット (I-format) とは機能的には似たものであるが、考え方の点では大きく違っている。E-format はあくまでも2オペランド一般形 (G-format) の派生形であり、ソースオペランドのサイズが8ビット固定、ディスティネーションオペランドのサイズが8/16/32/64ビットから選択となっている。つまり、異種サイズ間の演算を前提とし、destのサイズに合わせて8ビットのsrcがゼロ拡張または符号拡張される。一方、I-format は、特にMOV, CMPで頻度の多いイミディエートのパターンを短縮形にしたものであり、ソースとディスティネーションのサイズは等しい。

【0062】6-3-3. 第一オペランドはアドレス計算 (GA-format)

図22にその例を示す。

【0063】6-3-4. その他の2オペランド命令
図23にその例を示す。

【0064】6-4. ショートブランチ

図24にその例を示す。

【0065】6-5. その他

以上の外に図25に示すようなものがある。

【0066】7. アドレッシングモード

本発明装置のアドレッシングモードには、レジスタを含めて6ビットで指定する短縮形 (Sh) と、8ビットで指定する一般形 (Ea) がある。未定義のアドレッシングモードを指定した場合や、意味的に考えて明らかにおかしいアドレッシングモードの組み合わせを指定した場合には、未定義命令を実行した場合と同じく予約命令例外 (RIE) を発生し、例外処理を起動する。これに該当するのは、destinationがイミディエートモードの場合、アドレス計算の命令でイミディエートモードを使用した場合などである。

【0067】7-1. Pビット

本発明装置では、毎回のメモリアクセスに対応して1ビットのオプション機能指定ビットを割り当てることができるようになっており、このビットをPビットと呼ぶ。Pビットは、メモリアクセスに伴って何らかの別の意味を加えたい場合に使用するビットである。Pビットは、毎回のメモリアクセス毎に独立に指定する。したがって、レジスタ間接アドレッシング、アブソリュートアドレッシングなどの場合はオペランドに対応して一つのPビットを指定するが、付加モードを使用した多段間接のアドレッシングモードでは、その段数分だけのPビット

26

を指定することになる。Pビットの用途としては、タグのチェック、論理空間の切り替え、32ビットアドレッシングと64ビットアドレッシングの切り替えなどがあるが、これらはすべて将来の拡張用であり、現在の仕様ではPビットはreservedとなっている。命令フォーマットの説明では、Pビットの部分を'P'で表示してあるが、ここは必ず0にしておかなければならない。Pビットが0になっていなかった場合には、予約命令例外 (RIE) が発生する。Pビットに関する機能は〈〈LU〉〉仕様である。

【0068】7-2. フォーマット中で使われる記号
Rn レジスタ指定

P Pビット (0でなければならない)

mem[EA] EAで示されるアドレスのメモリ内容

以下点線で囲まれた部分は、拡張部を示す。

【0069】7-3. レジスタ直接
アセンブラ表記:

Rn

オペランド:

Rn

フォーマット: 図26に示す。

【0070】7-4. レジスタ間接
アセンブラ表記:

RRn

オペランド:

mem[Rn]

フォーマット: 図27に示す。

【0071】7-5. レジスタ相対間接
アセンブラ表記:

@(disp, Rn)

@(disp:16, Rn)

@(disp:32, Rn)

オペランド:

mem[disp + Rn]

フォーマット: 図28に示す。

なおdispは符号付きとして扱う。

【0072】7-6. イミディエート
アセンブラ表記:

#imm_data

オペランド:

imm_data

フォーマット: 図29に示す。

なおimm_dataのサイズは、オペランドサイズとして命令中で指定される。

【0073】7-7. アブソリュート
アセンブラ表記:

50

27

```
@abs
@abs:16
@abs:32
@abs:64    <<LX>>
```

オペランド:

mem[abs]

フォーマット: 図30に示す。

なお32ビットアドレッシングの時は、abs:16で指定したアドレスは32ビットに符号拡張される。また、64ビットアドレッシングの時は、abs:16, abs:32で指定したアドレスは64ビットに符号拡張される。

【0074】7-8. PC相対間接

アセンブラ表記:

```
@(disp.PC)
@(disp:16.PC)
@(disp:32.PC)
```

オペランド:

mem[disp + PC]

フォーマット: 図31に示す。

PC相対間接モードにおいて参照されるPCの値は、そのオペランドを含む命令の先頭アドレスである。したがって、例えば無限ループは

JMP @0.PC)

という命令によって実現される。付加モードにおいてPCの値が参照される場合にも、同じように命令先頭のアドレスをPC相対の基準値として使用する。

【0075】7-9. スタックポップ

アセンブラ表記:

@SP+

オペランド:

```
mem[SP]
SPをインクリメント
```

フォーマット: 図32に示す。

@SP+のモードでは、オペランドサイズだけSPをインクリメントする。例えば、本発明装置64で64ビットデータを扱う時には、SPが+8だけ更新される。B, Hのサイズのオペランドに対する@SP+の指定も可能であり、それぞれSPが+1, +2だけ更新される。ただし、スタックのアラインメントがくずれて速度低下の原因になるため、使用上は注意した方がよい。オペランドに対して@SP+のモードが意味を持たないものに対しては、予約命令例外(RIE)を発生する。具体的に予約命令例外(RIE)となるのは、writeオペランド、read-modify-writeオペランドに対する@SP+である。

【0076】7-10. スタックプッシュ

28

アセンブラ表記:

@-SP

オペランド:

```
SPをデクリメント
mem[SP]
```

フォーマット: 図33に示す。

@-SPのモードでは、オペランドサイズだけSPをデクリメントする。例えば、本発明装置64で64ビットデータを扱う時には、SPが-8だけ更新される。B, Hのサイズのオペランドに対する@-SPの指定も可能であり、それぞれSPが-1, -2だけ更新される。ただし、スタックのアラインメントがくずれて速度低下の原因になるため、使用上は注意した方がよい。オペランドに対して@-SPのモードが意味を持たないものに対しては、予約命令例外(RIE)を発生する。具体的に予約命令例外(RIE)となるのは、readオペランド、read-modify-writeオペランドに対する@-SPである。

20 【0077】7-11. レジスタ相対付加モード

オペランド:

```
Rn ==> tmp
付加モード処理
```

フォーマット: 図34に示す。付加モードについては、後の章でまとめて説明する。

【0078】7-12. PC相対付加モード

オペランド:

```
PC ==> tmp
付加モード処理
```

30

フォーマット: 図35に示す。

【0079】7-13. 絶対付加モード

オペランド:

```
0 ==> tmp
付加モード処理
```

フォーマット: 図36に示す。

【0080】7-14. FP相対間接

40 アセンブラ表記:

```
@(disp.FP)
@(disp:4.FP)
```

オペランド:

```
mem[d4 * 4 + FP]
(disp = d4 * 4)
```

フォーマット: 図37に示す。

d4は符号付きとして扱い、オペランドのサイズとは関係なく必ずd4を4倍して使用する。したがって、この

29

モードにより (FP-8*4) から (FP+7*4) までの4の倍数のメモリアドレスが参照可能である。アセンブラで記述する場合には、ディスプレースメントとして4倍した値の方を書く。このアドレッシングモードは〈〈L2〉〉である。本発明装置ではFP相対間接モードは実装しないので、このアドレッシングモードが指定された場合は、予約命令例外 (RIE) となる。このアドレッシングモードは短縮形で利用できないので、例えば、

```
MOV    @ (disp, FP), R1
```

といった場合に、

```
MOV:G.W @ (disp:4, FP), R1
```

```
MOV:L.W @ (disp:16, FP), R1
```

がともに4バイトとなり、コードの選択に曖昧さが生じるという問題点がある。このモードが〈〈L2〉〉となっているのは、このような理由による。このモードは、本発明装置64になって短縮形の割合が減った時に、有効に利用することを狙ったものである。@ (d4:4, FP)、@ (d4:4, SP) のモードでは、オペランドサイズにかかわらずd4を4倍して使用するため、8ビット、16ビット、32ビットのローカル変数をスタックフレーム上に混在した場合に@ (d4:4, FP)、@ (d4:4, SP) のモードを利用しようとすると、本発明装置がbig-endianである関係上、各変数のMSB側をワード境界に合わせて配置する必要がある。これによって特に問題が起きるわけではないが、注意が必要である。@ (d4:4, FP)、@ (d4:4, SP) モードを利用するためのローカル変数配置の例を図38に示す。

【0081】7-15. SP相対間接

アセンブラ表記:

```
@ (disp, SP)
```

```
@ (disp:4, SP)
```

オペランド:

```
mem[d4 * 4 + SP]
```

```
(disp = d4 * 4)
```

```
tmp + Rx * scale + d4 * 4 ==> tmp    [I=0, D=0 の時
tmp + Rx * scale + dispx ==> tmp      [I=0, D=1 の時
mem[tmp + Rx * scale + d4 * 4] ==> tmp [I=1, D=0 の時
mem[tmp + Rx * scale + dispx] ==> tmp [I=1, D=1 の時
```

基本フォーマット: 図40に示す。

30

フォーマット: 図39に示す。

d4は符号付きとして扱い、オペランドのサイズとは関係なく必ずd4を4倍して使用する。ただし、d4が負の値であった場合の動作は規定されていない。したがって、このモードにより (SP) から (SP+7*4) までの4の倍数のメモリアドレスが参照可能である。アセンブラで記述する場合には、ディスプレースメントとして4倍した値の方を書く。このアドレッシングモードは〈〈L2〉〉である。本発明装置ではFP相対間接モードは実装しないので、このアドレッシングモードが指定された場合は、予約命令例外 (RIE) となる。このモードも、@ (disp:4, FP) と同様に、本発明装置64になって短縮形の割合が減った時に、有効に利用することを狙ったものである。

【0082】7-16. 付加モードのフォーマット

複雑なアドレッシングも、基本的には加算と間接参照の組み合わせに分解することができる。したがって、加算と間接参照のオペレーションをアドレッシングのプリミティブとして与えておき、それを任意に組み合わせることができれば、どんな複雑なアドレッシングモードをも実現することができる。付加モードはこのような考え方にたったアドレッシングモードである。複雑なアドレッシングモードは、モジュール間のデータ参照やAI言語の処理系に特に有用である。ただし、本発明装置ではメモリ間接アドレッシングモードが多用された場合処理速度が低下する場合があるので、メモリ間接アドレッシングモードの使用に際しては十分な注意が必要である。付加モードの指定は、16ビットを単位としており、これを任意回繰り返す。1段の付加モードにより、

定数(displacement)の加算

インデックスレジスタのスケールと加算

スケールは×1、×2、×4、×8

メモリの間接参照

を行なう。N段の付加モードにより、N+1段までの間接参照ができる。

基本的な付加モードの処理:

31

32

E[=00 間接参照なし、付加モード継続
 $tmp + disp + Rx * Scale ==> tmp$
 E[=01 間接参照あり、付加モード継続
 $mem[tmp + disp + Rx * Scale] ==> tmp$
 E[=10 間接参照なし、付加モード終了
 $tmp + disp + Rx * Scale ==> \text{address of operand}$
 E[=11 間接参照あり、付加モード終了
 $mem[tmp + disp + Rx * Scale] ==> \text{address of operand}$

 M=0 <Rx> をインデックスとして使用
 M=1 特殊なインデックス
 <Rx>=0 インデックスを加算しない
 (Rx=0)
 <Rx>=1 PC をインデックスRxとして使用
 (Rx=PC)
 <Rx>=2 ~ reserved
 D=0 付加モード中の4ビットのd4を4倍してdispとし、
 これを加算する。d4は符号付きとして扱い、オペラ
 ンドのサイズとは関係なく必ずd4を4倍して使用す
 る。
 D=1 付加モードの拡張部で指定されたdispx (16/32/64
 ビット) をdispとし、これを加算する。拡張部のサ
 イズはd4フィールドで指定する。
 d4=0001 dispx は16ビット
 d4=0010 dispx は32ビット
 d4=0011 dispx は64ビット <<LX>>
 XX インデックスのスケール(scale = 1/2/4/8)
 S インデックスレジスタのサイズ
 S=0 <Rx> は32ビットを符号拡張
 S=1 <Rx> は64ビット <<LX>>
 P Pビット <<LU>>

・Pビットは付加モードの各段に入る。Pビットは「すべてのメモリ参照で独立に指定できるビット」となっている。

・間接参照をする場合としない場合を選択できる。間接参照しない段は、多段のベースレジスタ、インデックスレジスタの加算に用いる。(mem[R1+R2+R3] など) これは、ユーザレベルでrelocation base registerなどを導入したい時に使用することがある。

・インデックスレジスタのサイズ
64ビットアドレス使用時でも32ビットデータがかなりの頻度で出てくると予想されるため、付加モードの各段で32/64のサイズ切り替えができるようになって

いる。

・レジスタ相対間接の@ (disp: 64, Rn) やメモリ間接のアドレッシングモードも付加モードを使用して実現する。

・PCに対して×2、×4、×8のスケールリングを行なった場合には、その段の処理終了後の中間値(tmp)として、インプリメントに依存した不定値が入る。この付加モードによって得られる実効アドレスは予測できない値となるが、例外は発生しない。マニュアル等では、PCに対する×2、×4、×8のスケールリングの指定は行なわないように、注意しておく必要がある。

フォーマットのバリエーション: 図41, 図42に示す。

【0083】7-17. 付加モード仕様のレベル
付加モードの利用方法としては、普通の間接参照、オブジェクトコードのモジュール化のための外部変数のテーブル参照、AI向け命令の実行などがあり、このうち、AI向けの用途ではかなり多くの段数の間接参照を使うことがあるが、普通の用途では、3~4段までの間接参照で十分なことが多い。任意段数の付加モードが利用できれば、コンパイラの中で段数による場合分けが不要になるので、コンパイラの負担が軽減されるというメリットがある。多段の間接参照の頻度が非常に少ないとしても、コンパイラとしては必ず正しいコードを発生できなければならないからである。しかし、インプリメントの方から考えると、任意の段数を許して実行中の割り込みを受け付けるようにするのはかなり重くなるため、全体のバランスとしてある程度の段数制限をするのはやむを得ない。そこで、本発明装置としては4段(付加モード、基本フォーマットの4つ分)までの付加モードが利用できるものを〈L1〉仕様とし、任意段数の付加モードのインプリメントは〈L2〉仕様としてクラス分けする。〈L1〉仕様でも、5回のメモリ間接参照まで可能である。5段(5ハーフワード)以上の付加モードに対しては、予約命令例外(RIE)が起動される。ただし、フォーマット上は任意の段数が可能になっているので、将来はそのままのフォーマットで段数を拡張することができる。本発明装置では任意段数の付加モードを許す。ただし、本発明装置では付加モードをもちいてメモリ間接アドレッシングを多用した場合、処理速度が低下することがあるので注意を要する。特に第2オペランドで多段の付加モードが用いられた場合、付加モード処理中は割り込みが受け付けられない場合があるので注意を要する。また、本発明装置32でも浮動小数点を扱うことを考え、'×8'のスケールリングをインプリメントする。'×8'のスケールリングは〈LX〉仕様ではなく〈L1〉仕様である。

【0084】8. インプリメント関連事項

【0085】8-1. 仮想記憶のサポート

(本発明装置では仮想記憶のサポートは行わない。)

仮想記憶を実現するため、命令の実行途中で発生したページフォールトに対してうまく回復処理を行なう必要がある。本発明装置では、原則として命令再実行方式を採用する。命令再実行方式でページフォールトが起こった場合には、それまでに変更したレジスタ類をプロセッサがすべてもとに戻してから、ページインの処理ルーチンを起動する。したがって、処理再開後に命令の始めより再実行しても矛盾は生じない。命令再実行方式では、原則として実行途中の状態を保持する必要がないので、実現機構は比較的簡単である。また、本発明装置では、命令再実行のことを考慮し、処理の途中で副作用を残す命令やアドレッシングモード(オートインクリメントなど)を極力避けるようにしている。ただし、ページフォ

ールトからの再実行では余分なメモリアクセスが起こることがあり、OSで入出力装置を操作する場合などには注意が必要である。例えば、一般形の命令において、第一オペランドでI/Oのリードを行ない、第二オペランドでページフォールトを起こした場合、命令の再実行でもう一度I/Oのリードを行なうため、I/Oの種類によっては矛盾を起こす。したがって、リードによって副作用のあるI/Oをアクセスする場合は、その命令のもう一方のオペランドでページフォールトを起こさないように注意し、マニュアルにも明記する必要がある。具体的には、もう一方のオペランドが必ずレジスタか常駐ページであればよい。また、MOV命令などでソースオペランドとデスティネーションオペランドが一部重なっていた場合、単なる再実行では矛盾を生じることがあるので、この点に対する注意も必要である。

例：2バイトのデータを1バイトずらす。デスティネーションがページ境界にまたがる。図43で、MOV.H命令により[N-2:N-1]を[N-1:N]に移す場合、デスティネーションの書き込みバスサイクルは2回に分かれる。まず[N-2]のデータが[N-1]に書かれ、次に[元のN-1]が[N]に書かれるものとする。[N-1]への書き込みの際にpage M-1がフォールトを起こすと、ページイン後[N-2:N-1]→[N-1:N]を再試行しようとしても、N-1の内容が既に関書き変わっているため、矛盾を起こす。さらに、LDMのような複数のデータ転送を行なう命令でも、転送元と転送先が重なっていた場合に、命令の再実行で矛盾が生じないようにする必要がある。例えば、

LDM R6, (R6-R10)

の場合、R6, R7をロードした後でR8を読んだ時にページフォールトが起きると、再実行した時に既にR6が書き換わっており、本当に命令の最初から実行すると、矛盾を生じる。これを避けるためには、以下のような対策をとる必要がある。

・命令の最初でページフォールトが起きないことを確認する。

・ページフォールト時に転送中のアドレスを示すテンポラリ値をスタックにセーブする。(一種の命令継続実行方式)

・R6の初期値を記憶しておき、ページフォールト発生時にはこれをもとの値に戻す。STMやその他の命令についても同様である。なお、命令の再実行を矛盾なく行なうため、LDM, STM, LDC TXでは付加モードを禁止している。また、ENTER, EXIT, JRN Gでは、メモリアクセスを伴うようなアドレッシングモードをすべて禁止している。

【0086】8-2. プログラムによる命令の書き換え
ストアプログラム方式の計算機では、一般に、これから自分の実行する命令プログラム自体をプログラムによ

って書き換えることが可能である。しかし、命令のプリフェッチや命令キャッシュなどを持つ最近の高性能プロセッサでは、プログラムで命令を書き換える場合の動作を保証しようとすると、ハードウェアの負担が極めて大きくなる。また、この機能は必要性が少なく、ソフトウェアの教育上も好ましくない。したがって、本発明装置では、ソフトウェアによってこれから実行する命令コードの書き換えを行なうことは原則として禁止し、そのような場合には動作を保証しないものとしている。ただし、OSからユーザまで含めたシステム全体の動作を見ると、どこかでプログラムのロード～実行といった流れを含んでいるため、すべての場合にわたって「動作を保証しない」とするわけにはいかない。また、特殊な用途では、ユーザプログラムで命令コードを生成し、それを実行したいという場合もある。したがって、何らかの条件が満たされた時には、ソフトウェアによって書き換えられた命令コードの実行動作を保証する必要がある。そこで、本発明装置では、命令コードを書き換えたということをプロセッサに知らせる命令PIBを用意し、この命令を実行することにより、以後、書き換えられた命令コードの実行動作を保証することになっている。この命令は、これから実行すべき命令コードが、以前（リセット時あるいは前回のPIB命令実行時）から変更されている可能性があるということを、プロセッサに通知するために使用する。インプリメント上は、この命令によってパイプライン、命令キュー、命令キャッシュのページを行なうことになる。

【0087】9. EIT処理

本発明装置では、プログラムの実行の流れとは非同期に行なわれる処理を総称して、EIT処理と呼んでいる。EIT処理は、通常、例外処理や割り込み処理と呼ばれているものである。EIT処理には、次のようなものが含まれる。

・内部割り込み（リング間コール、トラップ）

システムコール発行などの際に、プログラマが意識して発生させる。その時に実行中のコンテキストとは関連がある。

・例外割り込み（例外）

一般の命令の実行中に、何らかのエラーが起った場合に発生する。その時に実行中のコンテキストとは関連がある。

・外部割り込み（割り込み）

10 外部からのハードウェア的な信号により発生する。その時に実行中のコンテキストとは全く関連がない。EITとはException（例外割り込み）、Interrupt（外部割り込み）、Trap（内部割り込み）の頭文字を合わせた名称である。EIT処理に関する詳細は付録9を参照のこと。

【0088】10. PSWの構成

本発明装置のPSW（Processor Status Word）は32ビットである。PSWの下位16ビット（PSH—Processor Status Halfword）はユーザプログラム用であり、ユーザプロセスから自由に操作可能である。PSWの上位16ビット（PSS—Processor Status halfword for System）はシステム用であり、ユーザプログラム（リング3）からは操作できない。PSHのうち、上位8ビットは各種モードの設定を行なう部分であり、PSM（Processor Status byte for Mode）と呼ぶ。また、PSHの下位8ビットは各種ステータスや演算結果の表示を行なう部分であり、PSB（Processor Status Byte）と呼ぶ。図44に示す。

【0089】10-1. PSSの構成

図45に示す。

37

38

'0' にreserved
 '1' を書き込もうとした場合には、予約機能例外(RFE)
)が発生する。

SM. RNG = 000	ring0 で外部割り込みスタックポインタ (SPI) 使用
SM. RNG = 001	reserved
SM. RNG = 010	reserved
SM. RNG = 011	reserved
SM. RNG = 100	ring0 でリング0 用スタックポインタ(SPO) 使用
SM. RNG = 101	reserved (ring1 用)
SM. RNG = 110	reserved (ring2 用)
SM. RNG = 111	ring3 でリング3 用スタックポインタ(SP3) 使用
	SM. RNGは<<LA>>
XA = 0	32ビットコンテキスト
XA = 1	64ビットコンテキスト
	<<LX>>
AT = 00	アドレス変換なし
AT = 01	アドレス変換あり (本発明装置標準のMMU 仕様)
AT = 10	アドレス変換なし、アドレスによるメモリ保護 <<L1R>>
AT = 11	reserved (Address Translation mode)
DB = 0	デバッグ中でないコンテキスト
DB = 1	デバッグ中コンテキスト
IMASK	外部割り込み、DI(Delayed Interrupt) を禁止する割 り込み優先度
IMASK = 0000	NMI (優先度0 のマスク不能割り込み) のみ受け付け
IMASK = 0001	優先度1 までマスク (結果的にNMI のみ受け付けとな る)
IMASK = 0010	優先度2 までマスク
...	IMASK の示す割り込みより優先度の高い割り込みのみ 受け付ける
IMASK = 1110	優先度14までマスク
IMASK = 1111	マスクしない

・本発明装置では、<<LA>>仕様として4レベルの 50 リング保護によるメモリ管理を行なう (付録参照)。本

39

発明装置では2レベルのリング保護によるメモリ管理を行なう。RNGフィールドは、現在プロセッサがどのリングにいるかという状態を示すものである。リング保護を行なわない場合にも、例えばスーパーバイザ、ユーザモードの切り換え用にこのフィールドを使用する。

・XAビットは、本発明装置32ではreservedであり、1を書き込もうとすると例外が発生する。

・トレースなどデバッグ関係の情報については、その詳細まで統一するのは難しいため、別の制御レジスタ(DCR-Debug Control Register)に分離している。ただし、デバッグ中がどうかを示す情報のみDBとしてPSWに入れる。

・本発明装置の外部割り込みは、低い優先度の方が大きな数字になる。外部割り込みの優先度は、0~6の7レベルであり、優先度0はマスク不能割り込み(NMI)である。

・キャッシュやMMUの制御情報は完全な統一が難しい*

「0」にreserved「1」を書き込もうとした場合には、予約機能例外(RFB)が発生する。

PRNG このリングに入る一つ前の状態のリング番号

PRNGは<<LA>>

P P-bit Error Flag <<LU>>

P-bit 機能の関連でエラーが起きたときにセットされ、他の場合にはクリアされる。

現在は0にreserved

F General Flag

高機能命令の終了要因の判定などに用いる。

X Extension Flag

多倍長計算用の桁上がりなどを示す。

V Overflow Flag

オーバーフローが発生したことを示す。

L Lower Flag

比較命令などにおいて、第一オペランドの方が小さいことを示す。(符号付き演算、符号なし演算とも)

M MSB Flag

演算結果のMSBが1であることを示す。

Z Zero Flag

演算結果が0になったことを示す。

・PRNGフィールドで「一つ前のリング」とは、「一つ外側のリング」あるいは、「そのリングにサービスを依頼したリング」を表わすものである。したがって、EIT発生時のPRNGの変化は、

PS W<RNG> ==> PSW<PRNG>

リターン時(REIT命令)でのPRNGの変化は、
スタック ==> PSW<RNG, PRNGを含む>

40

*ため、PSWとは分離している。

・AT(アドレス変換指定フィールド)をPSWに入れたことによって、コンテキスト毎にアドレス変換やメモリ保護の方法を変えたり、EIT処理ハンドラ実行中のみ一時的にアドレス変換を止めたりすることが可能になっている。

なお、LDC, REIT, LDCTX, EIT起動などによって、PSW中のAT(アドレス変換ビット)が00から01に変更された場合には、TLBやキャッシュのページが自動的に行なわれ、TLBや論理キャッシュの整合性が保証されるものとする。また、ATが01から00に変更された場合にも、キャッシュ(この場合は論理キャッシュ兼物理キャッシュ)の整合性が保証されるものとする。

【0090】10-2. PSHの構成

図46に示す。

となる。リターン時はRNGよりコピーするのではなく、必ずスタックより復帰する必要がある。常にRNG ≤ PRNGが成立する。PRNGは、ACS命令などでの参照を目的としたもので、実際のリング遷移はあくまでもRNGの情報を使用する。

・本発明装置以外のプロセッサでは、比較〜条件ジャンプ、といった命令の流れをとる場合に、符号付きと符号

41

なしの区別を比較命令ではなく条件ジャンプ命令で行なうのが普通である。例えば、符号なし整数の比較を

```
CMP    src1, src2
```

```
BLTS   next
```

Branch Lower Than(Signed)

で、符号付き整数の比較を

```
CMP    src1, src2
```

```
BLTU   next
```

Branch Lower Than(Unsigned)

で行なう。したがって、フラッグの表現する情報として、大小の区別のほかに、符号付きと符号なしの区別も必要である。しかし、本発明装置では、符号付きと符号なしの区別がCMP命令、CMPU命令といったように命令別になっており、条件ジャンプ命令は符号付きも符号なしも共通である。したがって、フラッグ構成を簡単にすることができる。

・通常のプロセッサで使用するCarry Flagは、符号なし整数の大小関係を表わすという意味と、多倍長演算の桁上がりを表わすという意味がある。しかし、後者に関しては本発明装置ではX_flagを使用するため、Carry Flagは整数の大小関係を表わすという意味でのみ用いられる。したがって、TRON

CHIPではこのフラッグを大小関係を表わすフラッグであると定義し、名前をL_flag(Lower Flag)としている。このフラッグは、符号なし演算の場合には従来のCarry Flagと同じ振る舞いをするが、符号付き演算の場合には従来のCarry Flagとは異なり、オーバーフローまで考慮した真の

・そのほか、ストリング命令やキューの命令の終了条件を示すためのF_flag(General Flag)とPビットのエラーを表現するためのP_flag(P-bit Error Flag)を設ける。P_flagは、現在の仕様では'0'にreservedとなっている。

・通常のプロセッサでは、シフト命令ではみだしたビットを入れるためにCarry Flagを用いているが、本発明装置ではCarry Flagの代わりにL_flagを実装しているため、はみだしたビットはX_flagに入れることにする。

【0091】10-3. フラッグの変化

例： @dest.B = 1 の時

```
SUB    #H'101.W, @dest.B ==> 演算結果1 - H'101
```

は0でないが、destが0になるのでZ_flagはセットされる。

```
CMP    #H'101.W, @dest.B ==> 演算結果1 - H'101
```

は0でないため、Z_flagはクリアされる。

42

加減算命令、比較命令、論理演算命令は2オペランド命令であり、

```
dest.op. src ==> dest
```

の形をとる。destとsrcのサイズが異なる場合には、小さいサイズの方が大きいサイズに合わせて符号拡張(ADDU, SUBU, CMPUではゼロ拡張)された上で演算され、演算結果がdestのサイズに変換されてからdestに格納される。CMP, CMPU, SUB, SUBUの場合、L_flagは、前の演算で第一オペランドの方が値が小さかったことを示す。符号なし演算CMPU, SUBUの場合には、L_flagは通常のプロセッサのCarry(Borrow)Flagと同じ意味になる。符号付き演算の場合には、L_flagは単なるM_flagのコピーとは異なり、オーバーフローまで含めた真の大小関係を表現する。ADD命令の場合には、L_flagは結果が負であることを示す。これも、単なるM_flagのコピーとは異なり、オーバーフローまで含めた真の正負を示す。ADDUの場合には、結果が必ず正になるため、L_flagは0となる。V_flagは、演算の結果がdestで指定されたサイズでは表現できなかったことを示す。つまり、演算結果がdestのサイズの符号付き整数(ADDU, SUBUでは符号なし整数)で表現できない時に、V_flagがセットされる。CMP, CMPU命令では、V_flagは不変である。X_flagは、多倍長の演算を行なう場合に、桁上りの状態を保持するために使用する。符号付き演算でも符号なし演算の時と同じような変化をする。これは、通常のプロセッサのCarry Flagとほぼ同じ意味であるが、X_flagを変化させる命令は加減算命令やシフト命令などに限られている。CMP命令とSUB命令、およびCMPU命令とSUBU命令のL_flagの変化は全く同じである。X_flagは、SUB, SUBU, SUBX命令では変化するが、CMP, CMPU命令では変化しない。MOV, MOVU, ADD, ADDU, ADDX, SUB, SUBU, SUBX命令の場合、M_flagとZ_flagは、演算結果をdestのサイズに変換した後の値を基準にして変化する。したがって、srcのサイズよりもdestのサイズの方が小さい時は、演算結果が0でなくてもZ_flagがセットされることがありうる。一方、CMP, CMPU命令の場合のZ_flagは、演算結果そのものの値を基準にして変化し、destのサイズには関係しない。

43

ADDX, SUBX命令のフラグの変化は、多少変則的になっている。これは、符号なし整数の拡張演算にも符号付き整数の拡張演算にも対処するためである。この*

L_flag 符号付き演算としての大小関係(SUBX)、正負(ADDX)を示す

。

V_flag 符号付き演算としてのオーバーフローを示す。

X_flag ADDXの場合はdest + src + X_flagの演算におけるdestのサイズからの桁上がり、SUBXの場合は、dest - src - X_flagの演算におけるdestのサイズからの桁下がりを変わす。ただし、いずれの場合も、srcのサイズがdestのサイズよりも小さい場合には、srcが符号拡張される。SUBXにおいて、srcとdestのサイズが等しい場合には、結果的に、X_flagが符号なしデータとしての比較結果を変わすことになる。

ADDX, SUBXで異種サイズ間の演算を行なう場合には、サイズの短い方が符号拡張される。しかし、符号拡張後の値を符号付きの数とみて演算するか、符号なしの数とみて演算するかはフラグによって異なる。MOV命令、MOVU命令および論理演算命令では、X_flag, L_flagは変化しない。論理演算命令では、V_flagも変化しない。各命令に対応したフラグの変化は、命令セットの説明の中に示されている。'☆'は要注意箇所である。

【0092】11. 命令セットの記述について

【0093】11-1. 記述形式の概要

〔ニモニック〕その命令の名前(ニモニック)を示す。

〔命令の機能〕その命令の機能の概要を示す。

〔命令オプション〕その命令で利用できる命令オプションの種類を示す。命令オプションは、命令の機能の細かい点を変更するために用いるものであり、アセンブラ表記では' / x x x 'により記述する。

〔命令ビットパターンとアセンブラ表記〕命令のビットパターン、そのアセンブラ表記、利用できるサイズの種類などを示す。本発明装置では、一つの命令ニモニックに対して一般形や短縮形といった複数の命令フォーマットが存在する場合があり、それぞれ利用できるアドレッシングモードやサイズが異なっている。この項では、命令フォーマット別にそういった内容を明らかにする。

〔フラグ変化〕命令実行後のステータスフラグ(P, S, B)の変化を示す。

〔解説〕その命令の機能を解説する。なお、説明の中で現れるアセンブラニモニックの詳細については、巻末の付録を参照のこと。

【0094】11-2. 命令ビットパターンとアセンブラ表記

命令ビットパターンとアセンブラ表記の部分は、フォーマット別ニモニック、オペランド名、オペランドフィー

44

*場合、条件ジャンプ命令のニモニックとの対応がうまくとれなくなるが、拡張演算自体が頻度も少なく、変則的な面を持っているので、やむを得ない。

ルド名、命令ビットパターンから成る。

記述例：図47に示す。

20 AND: G—フォーマット別ニモニック

説明を行なう命令ビットパターンのフォーマット別ニモニック(付録参照)を示す。

src, dest—オペランド名

その命令の機能を説明するために使用する変数である。

この変数は、「命令の機能」「解説」で参照される。ここで記述されたオペランドの順番が、そのままアセンブラにおけるオペランドの順番になる。

EaR, EaM—オペランドフィールド名

オペランドフィールド名は、ビットパターンとの対応、

30 使用できるオペランドサイズやアドレッシングモード、メモリアクセス方法、その他の制約事項などの情報をまとめて表わすものである。オペランドフィールド名を表わす文字とその意味との間には一定の原則を設けておき、いろいろな意味を簡潔に表現できるようにしている。

枠でかこまれた部分—命令ビットパターン

命令ビットパターン中では、オペランドフィールドやサイズ指定フィールドの位置、命令のオペコードなどを示す。' * 'で示されるビットは、don't careのビットである。このビットの0/1は、命令デコードには影響しない。' - ', ' + ', ' = ', ' # 'で示されるビットは、現在のところ、命令機能やオペランドの区別には使用されていないビットである。ただし、ユーザプログラムでは' - ', ' = 'の部分には0を、' + ', ' # 'の部分には1を入れておかなければならない。' - 'のビットが0でない場合や' + 'のビットが1でない場合は、予約命令例外(RFE)となる。' = 'のビットが0でない場合や' # 'のビットが1でない場合は、単に無視される。つまり、ハード的には' * ', ' = ', ' # 'は同等の意味を持つ。しかし、将

45

来の拡張のために、ユーザ向けのマニュアルには、
 =', '#' を '0', '1' としておくように明記し
 ておかなければならない。

【0095】11-3. フィールド名

命令ビットパターン中には、オペランドフィールドのほ*

RR read アクセスを行なうオペランドのサイズ指定
 WW write アクセスを行なうオペランドのサイズ指定
 MM read-modify-write アクセスを行なうオペランドのサイズ指
 定
 BB ビット操作命令でのメモリアクセスサイズ
 XX 上記以外の一般的なサイズ指定
 特にレジスタサイズの指定を行なう場合
 SS 上記以外の一般的なサイズ指定
 ディスプレースメントのサイズ、CMP の第二オペランド、暗
 黙でオペランドを指定するストリング命令、暗黙でスタック
 を指定する MOVA:U 命令などに使用
 必ず同じ文字（大文字）を反復する。ただし、32ビットと64
 ビットの指定しかできない場合には、このうちの一文字のみ
 を使う。

46

*かに、オプションフィールド、サイズ指定フィールドが
 ある。本発明装置で使用しているオプションフィールド
 名、サイズ指定フィールド名には、次のようなものがあ
 る。

・オプションフィールド名

命令オプションの指定を行なうオプションビットの名前
 としては、主として小文字を使う。（Pビット関係を除
 く）

※オプションフィールド名には、以下に示すようなものが
 ある。いずれの場合にも、最初に記述した方（オプショ
 ン値が0, 00...の方）がアセンブラでのデフォルト
 になる。

cccc Bcc. TRAP/cc での条件指定
 eeee ストリング命令、QSCH命令での終了条件指定
 P, Q.. P ビット指定 (Q.. は、P ビットの必要なオペランドが複
 数の場合)
 b /R=0./B=1 (BSCH, BVSCCH, BVMAP, BVCPY, SCMP, SMOV, QSCH)
 r /R=0./R=1 (SSCH)
 c /N=0./S=1 (CHK) -- CHK, change index value の'c'
 d /O=0./I=1 (BSCH, BVSCCH) -- dataの'd'
 m /NM=0./MR=1 (QSCH) -- maskの'm'
 p /AS=0./SS=1 (PTLB, PSTLB, LDATE) -- PTLB, sPeci fic
 space の'p'
 ttt /PT=000./ST=001./AT=110,
 /reserved=010 ~101, 111 (PSTLB, LDATE, STATE)
 xx /LS=00./CS=01 reserved=10, 11 (LDCTX, STCTX)

以上の項目に当てはまらないフィールド名は、オペラン
 ドフィールド名を示すものになる。できるだけ、同じ文
 字が複数の意味を表わさないようにしている。

【0096】11-4. オペランドフィールド名

オペランドフィールド名を表わす文字には、以下のよう
 な意味を持たせている。オペランドフィールド名はこれ 50

らの文字の組み合わせによって構成されるため、フィー
 ルド名だけで、使用できるアドレッシングモード、オペ
 ランドサイズ、アクセス方法などの情報を得ることがで
 きる。

47
・基本となるアドレッシングモード

Ea	8ビットの一般形アドレッシングモードを使用
Sh	6ビットの短縮形アドレッシングモードを使用
#	リテラル
#i	イミディエート
#d	ディスプレースメント
Rg	レジスタ
LI	レジスタリスト (LDM 用)
Ls	レジスタリスト (STM 用)
Ln	レジスタリスト (ENTER 用)
Lx	レジスタリスト (EXITD 用)

48
*【0097】・アクセス方法

一部の基本アドレッシングモードでは、以下のようなアクセス方法がデフォルトとして決まっている。この場合には、特にアクセス方法を示す文字を付けない。

#, #i, #d	命令空間からのread
Ls, Ln	レジスタのread
LI, Lx	レジスタへのwrite

その他の基本アドレッシングモードについては、以下に

10 示す文字を使用してアクセス方法を示す。

R	read
W	write

*
M read-modify-write

なお、フィールド名を短縮するため、RgR をRRに、RgW をRWに、RgM をRMに省略することがある。(BF命令、CSI 命令)

A アドレス計算のみを行なう。

f ビットオフセットとの組み合わせによって実際に操作を行なうメモリアドレスが決まる。(R, Mのサフィックス)

例: ビット操作命令

fq ビットオフセットが付くが、ビットオフセットはバイト境界を越えない。アクセスすべきアドレスは、オフセットを見なくても確定している。(R, M のサフィックス)

例: 短縮形のビット操作命令

bf ビットオフセットやビットフィールド幅との組み合わせによって実際に操作を行なうメモリアドレスの範囲が決まる。(R, M のサフィックス)

例: 固定長ビットフィールド操作命令

q キュー命令による複雑なアクセスを行なう。(他のアクセス方法のサフィックス)

例: QINS, QDEL 命令

i バスのインタロックによるアクセスを行なう。(M のサフィックス)

% 制御空間、物理空間などの特殊空間のアクセスを行なう。(R, W, M のサフィックス)

d 2つのデータ(double)に対する操作を行なう。(R のサフィックス)

例: CHK 命令

m 複数のデータ(multiple)に対する操作を行なう。(R, W のサフィックス)

例: LDM, STM 命令

・アドレッシングモードに対する制限

基本アドレッシングモードとアクセス方法が決まると、自動的にアドレッシングモードに対する制限 (E a Wに

に対するイミディエートモードの禁止など)が決まる。ただし、それ以外に命令特有の制限事項がある場合には、以下の文字を後ろに付ける。

- 49
- !! イミディエートモードの禁止
例: CMP 命令の第二オペランド
- !M メモリ対象アドレッシングモードの禁止
例: ENTER:G 命令のlocal オペランド
- !A 付加モードの禁止
例: LDCTX 命令のctxaddr オペランド
- !S スタックポップ、スタックプッシュモードの禁止
例: QDEL命令のdestオペランド

50

*【0098】・サイズ指定
サイズ指定は、原則として以下に示すフィールドによって行なう。

*10

アクセス方法がR
RRフィールド

アクセス方法がW
WWフィールド

アクセス方法がM
MMフィールド

アクセス方法がR!L, R!M, R2
SSフィールド

アクセス方法が*
BBフィールド

ただし、これはメモリ操作のアクセスサイズを意味する。

アクセス方法がA

サイズは指定されない。

【0099】これより例外がある場合には、以下の文字を付け加えることにより区別する。原則として、数字と小文字が固定サイズを表わし、大文字は可変サイズを表

わす。例えば、'w' は32ビット（ワード）固定のサイズを示すのに対して、'W' はWWフィールドによりサイズが指定されることを示す。

- 51
w オペランドサイズは必ず32ビット
例: MUL:R 命令
- h オペランドサイズは必ず16ビット
例: WAIT命令
- b オペランドサイズは必ず8ビット
例: MOV:E 命令のsrc
- S8 オペランド(ディスプレースメント)のサイズは、SSフィールドにより指定される。ただし、このオペランド指定フィールドを使うのは、SS=00 (8bit 指定)の時に限られる。それ以外の場合には、拡張部によってオペランドが指定され、このフィールドは無視される。(0にしておくこと)
例: BF:I命令のsrc
- S オペランド(ディスプレースメント)のサイズは、SSフィールドにより指定される。
例: BRA:G 命令
- R オペランドサイズは、もう一方のオペランドのサイズと共通にRRフィールドにより指定される。
例: CMP:I 命令
- W オペランドサイズは、もう一方のオペランドのサイズと共通にWWフィールドにより指定される。
例: MOV:I 命令
- M オペランドサイズは、もう一方のオペランドのサイズと共通

52

53

54

にMMフィールドにより指定される。

例：Iフォーマットの命令

- L オペランドサイズとして8ビットまたは16ビットを指定するためのビットパターンが割り当てられていないため、32ビットまたは64ビットのオペランドのみが指定できる。

サイズ指定は、RR.WW.MM.BB フィールドではなく、R.M.W.B フィールドにより行なわれる。

- P ポインタを扱うため、命令中ではサイズ指定を行なわない。実際のサイズ指定は、P ビットまたはモード（PSW 中のXAビット）等によって行なわれる。

例：QINS.QDEL 命令

- X オペランドサイズは、XXフィールドにより指定される。

例：ACB.SCB 命令のxreg

- Xw オペランドサイズは、Xフィールドにより他のオペランドと共通に指定される。これは、BF命令のwidth 指定用である。

- Xs オペランドサイズは、Xフィールドにより他のオペランドと共通に指定される。これは、BF命令のsrc 指定用である。

- Xd オペランドサイズは、Xフィールドにより他のオペランドと共通に指定される。これは、BF命令のdest指定用である。

- C オペランドサイズは、RRフィールドにより他のオペランドと共通に指定される。これは、CSI 命令の比較値指定用である。

- 3 3 ビットのリテラル

- 4 4 ビットのリテラル

例：TRAPA 命令

- 6 6 ビットのリテラル

- 8 8 ビットのディスプレースメント

例：BRA:8 命令

- 16 16ビットのディスプレースメント

例：MOVA:R命令

なお、ストリング命令などの高機能命令において、命令によって暗黙に指定されるオペランドのサイズを指定す

る場合には、フィールド名としてSSを使用する。任意長ビットフィールド命令では、Xも使用される。

【0100】・その他

55 Z リテラルで、ビットパターン⁵⁶の0 をオペランド値の0(zero) に対応させる場合を示す。ビットパターンとオペランド値との対応は、以下ようになる。(N はリテラルのビット数)

0...000	0
0...001	1
0...010	2
...	
1...110	2^{N-2}
1...111	2^{N-1}

例: BTST:Qのoffset

n リテラルで、ビットパターン⁵⁶の0 をオペランド値の 2^N に 対応させる場合を示す。ビットパターンとオペランド値との 対応は、以下ようになる。(N はリテラルのビット数)

0...000	2^N
0...001	1
0...010	2
...	
1...110	2^{N-2}
1...111	2^{N-1}

例: MOV:Q のsrc

c リテラルで、ビットパターンが2の補数(complement)を現わ

す場合を示す。ビットパターンとオペランド値との対応 は、以下ようになる。(Nはリテラルのビット数)

0...000	-2^N
0...001	$-(2^{N-1})$
0...010	$-(2^{N-2})$
...	
1...110	-2
1...111	-1

例: SHA:C, SHL:C で右シフトの場合のシフトカウント

1, 2, ... 一つの命令の中で、同じアクセス方法を持 *

EaR, ShR

@-SPは利用できない。

EaW, ShW

#imm_data, @SP+ は利用できない。

EaM, ShM

#imm_data, @-SP, @SP+は利用できない。

EaA

@SP+, @-SP, Rn, #imm_dataは利用できない。

*つオペランドが複数存在した場合に、それらを区別する ために使用する。なお、サイズに関する種々の制限事項 のうち、命令機能に大きな関係を持つものについては、 30 オペランドフィールド名やサイズ指定フィールド名では なく、各命令の説明のところでその制限を示す。これに は、シフトカウントで8ビット以外のサイズを指定した 場合や、異種サイズ間の論理演算などが含まれる。

【0101】11-5. アドレッシングモードに関する 制限

オペランドフィールド名のうち、次のものは、使用でき るアドレッシングモードに制限が設けられている。

このほか、各命令の説明のところでもアドレッシングモ ードに関する制限が述べられている。

【0102】11-6. 解説に関する注意

スタック操作の命令では、TOSによりスタックトップ を示している。↑TOSはスタックからのポップ、↓T OSはスタックへのプッシュである。

57

58

2オペランドの基本命令(MOV, MOVU, ADD, ADDU, ADDX, SUB, SUBU, SUBX, AND, OR, XOR, CMP, CMPU)では、次のような記法でそのオペレーションを説明している。

dest(src2)のサイズ(ビット数)をdで、src(src1)のサイズ(ビット数)をsで表わし、src(src1), dest(src2)をビット分解した値をD0, D1, ..., Dd-1, S0, S1, ..., Ss-1で表わす。したがって、

$$\text{dest}(\text{src2}) = [D0, D1, \dots, Dd-2, Dd-1]$$

$$\text{src}(\text{src1}) = [S0, S1, \dots, Ss-2, Ss-1]$$

と書ける。[...]は2進数による表示を、'.'は各桁の区切りを意味する。この時、演算結果によってdestに設定される値を

$$\text{dest.op. src} = \text{result}$$

$$= [R0, R1, \dots, Rd-2, Rd-1]$$

で表わす。MOV, MOVU, CMP, CMPU 以外の命令では、resultがdestにセットされる。またs > dの場合は、一般に演算結果の下位ビットのみをdestに設定することになる。この時、演算結果の上位ビットをカットする前の値を

$$\text{result} = [F0, F1, \dots, Fs-2, Fs-1]$$

で表わす。Rのビット数はd、Fのビット数はsである。

ビット列[...]を符号付きの2進数と解釈した場合には、そのビット列の表現する値をS[...]で表わし、符号なしの2進数と解釈した場合には、そのビット列の表現する値をU[...]で表わす。また、そのビット列を符号付きPACKED形式10進数と解釈した場合には、そのビット列の表現する値をSD[...]で表わし、符号なしPACKED形式10進数と解釈した場合には、そのビット列の表現する値をUD[...]で表わす。さらに、

【0103】

【数5】

... は真偽反転を、... は累乗を表わす。

【0104】

同じように、固定長ビットフィールド命令の説明では、

$$\text{bitfield} = [B0, B0+1, \dots, B0+w-2, B0+w-1]$$

といった記法でビット単位の詳細なオペレーションを説明している。

。

略記法として、

$$[Sn, Sn+1, \dots, Sm-2, Sm-1] \text{ を}$$

$$[Sn \sim m-1]$$

で表わし、

$$[S0, S1, \dots, Sd-2, Sd-1]$$

59

60

= [S0 ~s-1]

を単に

[S]

で表わすことがある。[D], [R], [B], [F] についても同様である。

【0105】12. 「本発明装置」の命令セット

【0106】12-1. データ転送命令

〔ニモニック〕

MOV src, dest

10

〔命令の機能〕

src ==> dest ..

データの移動と符号拡張

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図48に示す。

〔フラッグ変化〕図48の最下部に示す。

〔解説〕ソースオペランドsrcをデスティネーションオペランドdestに転送する。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小

20

[MOV]

:Z src = 0

:Q $1 \leq \text{src} \leq 8$:E $-128 \leq \text{src} \leq 127$

:I src は任意

:G src は任意

[MOVU]

:B $0 \leq \text{src} \leq 255$

:G src は任意

ADD, SUB, CMP 命令も同様である。

さい時は、ソースが符号拡張される。デスティネーションの方がサイズが小さく、ソースの値がデスティネーションのサイズの符号付き整数として表現できない時は、V__flagがセットされる。MOV: Zはいわゆるclear命令であるが、動作やフラッグ変化が同じであるため、MOVの短縮形の一つとして扱っている。MOV, ADD, MOV, CMP命令は符号付きの演算を行なう命令であるが、MOV: Q, ADD: Q, SUB: Q, CMP: Qで利用できるリテラルの範囲は1~8 (オペランドフィールド名#3n) となっており、正の範囲しか含んでいないので、注意が必要である。MOV, MOVU命令でsrcがイミディエート値である場合に、そのイミディエート値と利用できるフォーマットとの関係をまとめると、次のようになる。

61
($d \geq s$ の時)
 $\{S0, S1, \dots, Ss-2, Ss-1\} ==>$
 $\{S0, S0, \dots, S0, S0, S1, \dots, Ss-2, Ss-1\} ==>$
 $S0, S1, \dots, Ss-2, Ss-1\}$
 $d-s$ ビットだけ符号拡張
 $\{R0, R1, \dots, Rd-s+1, Rd-s, Rd-s+1, \dots, Rd-2, Rd-1\}$ (destに設定される)

($d < s$ の時)
 $\{S0, S1, \dots, Ss-d-1, Ss-d, Ss-d+1, \dots, Ss-2, Ss-1\} ==>$
 $\{Ss-d, Ss-d+1, \dots, Ss-2, Ss-1\} ==>$
 $S0, S1, \dots, Ss-d-1$
 $s-d$ ビットがカットされる。
 $\{R0, R1, \dots, Rd-2, Rd-1\}$ (destに設定される)
 $M_flag \quad R0$
 $Z_flag \quad [R0 \sim d-1] = 0$
 $V_flag \star S[S] < -2^{(d-1)} .or. S[S] \geq +2^{(d-1)}$
 つまり、 $d \geq s$ であればクリアされ、 $d < s$ であれば、
 $S0 = S1 = \dots = Ss-d-1 = Ss-d (=R0)$
 の時クリア、それ以外の場合にセットとなる。

【0107】 [プログラム例外]

・予約命令例外

- ・RR='11' のとき
- ・WW='11' のとき
- ・EaR, ShR が@-SPのとき
- ・EaW, ShW が#imm_data, @SP+ のとき

[ニモニック]

MOVU src, dest

[命令の機能]

zex(src) ==> dest

データの移動とゼロ拡張

($d \geq s$ の時)

$\{S0, S1, \dots, Ss-2, Ss-1\} ==>$
 $\{0, 0, \dots, 0, S0, S1, \dots, Ss-2, Ss-1\} ==>$

62

[命令オプション] なし

[命令ビットパターンとアセンブラ表記] 図49に示す。

[フラッグ変化] 図50に示す。

[解説] ソースオペランドsrcをデスティネーションオペランドdestに転送する。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースがゼロ拡張される。デスティネーションの方がサイズが小さく、ソースの値がデスティネーションのサイズの符号なし整数として表現できない時は、V_flagがセットされる。

63

d-s ビットだけゼロ拡張

[R0, R1, ..., Rd-s+1, Rd-s, Rd-s+1, ..., Rd-2, Rd-1] (destに設定される)

(d < s の時)

[S0, S1, ..., Ss-d-1, Ss-d, Ss-d+1, ..., Ss-2, Ss-1] ==>

[Ss-d, Ss-d+1, ..., Ss-2, Ss-1] ==>

S0, S1, ..., Ss-d-1

s-d ビットがカットされる。

[R0, R1, Rd-2, Rd-1 (destに設定される)

M_flag R0

Z_flag [R0~d-1] = 0

V_flag☆ U[S] ≥ +2 ^ d

つまり、d ≥ s であればクリアされ、d < s であれば、

S0 = S1 = ... = Ss-d-1 = 0

の時クリア、それ以外の場合にセットとなる。

【0108】 [プログラム例外]

・予約命令例外

・RR='11' のとき

・WW='11' のとき

・EaR が@-SPのとき

・EaW が#imm_data, @SP+ のとき

[ニモニック]

PUSH src

[命令の機能]

push to stack

スタックにプッシュ

[命令オプション] なし

[命令ビットパターンとアセンブラ表記] 図51に示す。

[フラッグ変化] 図52に示す。

[解説] ソースオペランドsrcをスタックにプッシュする。この命令は、MOV *, @-SPの短縮形と考えることもできるが、フラッグ変化をしないこと、およびPOPとの対称性により、別命令となっている。src/EaRLで指定されるアドレッシングモードでは、@SP+のモードは使用できない。これは、POP命令のdest/EaWLで@-SPのモードが使用できないのに合わせたものである。PUSH SPなど、srcオペランドにSPを含む場合の命令動作規定については、付録12を参照のこと。

【0109】 [プログラム例外]

・予約命令例外

・R='1' のとき

・EaRLが@SP+, @-SP のとき

[ニモニック]

64

POP dest

20 [命令の機能]

pop from stack

スタックからポップ

[命令オプション] なし

[命令ビットパターンとアセンブラ表記] 図53に示す。

[フラッグ変化] 図54に示す。

[解説] スタックからポップした値をdestに転送する。この命令は、MOV @SP+, *の短縮形と考えることもできるが、srcにSPを含んだ場合の動作がMOV @SP+とは異なること、およびフラッグ変化をしないこと、によって別命令となっている。dest/EaWLで指定されるアドレッシングモードでは、@-SPのモードを使用することは禁止されており、指定した場合には予約命令例外RIEとなる。これは、POP @-SPという命令を実行した場合に、SP更新がいつ行なわれるかという点について誤解を生じやすいためである。POP SPなど、destオペランドにSPを含む場合の命令の動作規定については、付録12を参照のこと。

40 【0110】 [プログラム例外]

・予約命令例外

・W='1' のとき

・EaWLが#imm_data, @SP+, @-SPのとき

[ニモニック]

LDM src, reglist

[命令の機能]

load multiple registers

複数レジスタのロード

50 [命令オプション] なし

〔命令ビットパターンとアセンブラ表記〕図55に示す。

〔フラッグ変化〕図56に示す。

〔解説〕複数のレジスタをメモリからロードする。ロードするレジスタはビットマップ `reglist/LIRL` (レジスタリスト) で指定する。LIRLは、EaRmLの拡張部よりも後に置かれる。ロードするレジスタリスト (`reglist`) のビットマップ指定は、図57に示すように行なう。EaRmLで@SP+のアドレッシングモードを指定した場合は、小さい番号のレジスタから順にポップされ、SPはロードしたレジスタ数の4倍 (または8倍) だけ増加する。それ以外のアドレッシングモードを指定した場合は、得られた実効アドレスがレジスタにロードすべきメモリデータの先頭を指す。いずれの場合にも、メモリ中では小さい番号のレジスタの方が低いアドレスに置かれる。ロードするレジスタのビットマップのフォーマットは、BSCH/F, BVSC H/F命令で使用する回路 (次に出現する'0'または'1'のビットをMSB方向にサーチする回路) と同じ回路によって、次に転送するレジスタを見付けられるように決めたものである。したがって、LDM @SP+の場合は小さな番号のレジスタから転送するためにレジスタ番号の小さな方がMSB側となっている。それ以外のアドレッシングモードの場合にも、レジスタ退避ブロックの先頭アドレスを実効アドレスとしているため、やはりレジスタ番号の小さい方から転送するのがよく、LDM @SP+と同じフォーマットになる。なお、これらのフォーマットはレジスタの転送順序まで考えて決めたものであり、ハードウェア資源が少ない場合には、ここで説明したような転送順序にするのが最適と考えられる。しかし、実際の転送の順序は「本発明装置」で規定されたものではなく、インプリメント側の自由である。EaRmLのアドレッシングモードでは、@-SP、レジスタ直接モードRn、イミディエートモード#imm_data、付加モードの指定はイリーガルとする。付加モードを禁止するのは、LDMやSTMによって退避、復帰したレジスタやレジスタ退避エリアと、付加モードで使用するレジスタやメモリの間にオーバーラップがあった場合に、命令の再実行が難しくなるためである。レジスタリストがオール0の時は、何もせずに命令を終了する。(特にエラーとはしない)

LDM @SP+でレジスタリストにSPが含まれる場合の動作規定については、付録12を参照のこと。

【0111】〔プログラム例外〕

・予約命令例外

・R='1' のとき

・EaRmL がRn, #imm_data, @-SP, 付加モードのとき

〔モニタック〕

STM reglist, dest

〔命令の機能〕

store multiple registers

複数レジスタのストア

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図58に示す。

〔フラッグ変化〕図59に示す。

〔解説〕複数のレジスタをメモリにセーブする。セーブするレジスタはビットマップ `reglist/LsWL` (レジスタリスト) で指定する。LsWLは、EaWmLの拡張部よりも後に置かれる。ストアするレジスタリスト (`reglist`) のビットマップ指定は、EaWmLが@-SPモードの時、図60に示すように、またその他のモードの時、図61に示すように行なう。EaWmLに@-SPのアドレッシングモードを指定した場合は、大きい番号のレジスタから順にプッシュされ、SPはセーブしたレジスタ数の4倍 (または8倍) だけ減少する。それ以外のアドレッシングモードを指定した場合は、得られた実効アドレスがレジスタをセーブすべきメモリ領域の先頭を指す。いずれの場合にも、メモリ中では小さい番号のレジスタの方が低いアドレスに置かれる。このフォーマットは、BSCH/F, BVSC H/F命令で使用する回路 (次に出現する'0'または'1'のビットをMSB方向にサーチする回路) と同じ回路によって、次に転送するレジスタを見付けられるように決めたものである。したがって、STM @-SPの時は大きな番号のレジスタから転送するためにレジスタ番号の大きな方がMSB側となる。それ以外のアドレッシングモードの場合には、レジスタ退避ブロックの先頭アドレスを実効アドレスとしているため、レジスタ番号の小さい方から転送するのがよく、レジスタ番号の小さな方がMSB側となっている。なお、これらのフォーマットはレジスタの転送順序まで考えて決めたものであり、ハードウェア資源が少ない場合には、ここで説明したような転送順序にするのが最適と考えられる。しかし、実際の転送の順序は「本発明装置」で規定されたものではなく、インプリメント側の自由である。EaWmLのアドレッシングモードでは、@SP+, レジスタ直接モードRn、イミディエートモード#imm_data、付加モードの指定はイリーガルとする。付加モードを禁止するのは、LDMやSTMによって退避、復帰したレジスタやレジスタ退避エリアと、付加モードで使用するレジスタやメモリの間にオーバーラップがあった場合に、命令の再実行が難しくなるためである。LDM, STM命令では、転送しないレジスタに対するメモリ領域は割り当てない。例えば、

STM.W (R1, R3, R9), @-SP

の場合は次のような動作を行なう。(ただし、命令実行前のSP値をinitSPとする。)

67
R9 ==> mem[initSP - 4]
R3 ==> mem[initSP - 8]
R1 ==> mem[initSP - 12]
initSP - 12 ==> SP

レジスタリストがオール0の時は、何もせずに命令を終了する。(特にエラーとはしない)

STM @-SPでレジスタリストにSPが含まれる場合の動作規定については、付録12を参照のこと。

【0112】〔プログラム例外〕

・予約命令例外

・W='1' のとき

・EaWnL がRn, #imm_data, @SP+, 付加モードのとき

〔ニモニック〕

MOVA srcaddr, dest

〔命令の機能〕

address of src ==> dest

実効アドレスを得る

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図62に示す。

〔フラッグ変化〕図63に示す。

〔解説〕ソースオペランドの実効アドレスをデスティネーションオペランドに転送する。この命令のオペレーションそのものは、MOV命令などで代用可能であるが、高級言語での左辺値のアドレス計算やポインタ演算にすなわち使用できること、アドレス計算用の回路を使用するため、より高速な演算が期待できること、により別命令となっている。短縮形の

MOVA:R @(<disp:16, Rs>).Rd

は、実質的には3オペランド加算命令

Ra + disp:16 -> Rd

となるが、フラッグ変化をおこさないためMOVA命令に分類されている。srcaddrにPC相対間接モードを指定し、PC相対のディスプレースメントを0とした場合には、現在のPC値、つまりMOVA命令の先頭アドレスをdestに格納することになる。また、PC相対のディスプレースメントとしてMOVA命令の命令長を指定した場合には、MOVA命令の次の命令のアドレスをdestに格納することになる。これらの機能は、ユーザプログラムのレベルでコルーチン処理を行なう時に有効である。アセンブラでは、<オペレーション>またはdest側でサイズ指定を行なう。srcaddr側はアドレス計算のみなので、サイズの指定はしない。EaAで指定されるアドレッシングモードでは、イミディエート、@SP+, @-SPのモードは使用できない。

【0113】〔プログラム例外〕

68

・予約命令例外

・+='0' のとき

・W='1' のとき

・EaA がRn, #imm_data, @SP+, @-SPのとき

・BaW が#imm_data, @SP+ のとき

〔ニモニック〕

PUSHA srcaddr

〔命令の機能〕

10 push address to stack

実効アドレスをスタックにプッシュ

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ記法〕図64に示す。

〔フラッグ変化〕図65に示す。

〔解説〕ソースオペランドsrcaddrの実効アドレスをスタックにプッシュする。この命令は、MOVA *, @-SPの短縮形と考えることもできるが、実行速度の向上や、MOV命令~PUSH命令の区別との対応をとるために、別命令となっている。srcにSPを含んだ場合の動作規定については、付録12を参照のこと。

【0114】〔プログラム例外〕

・予約命令例外

・S='1' のとき

・EaA がRn, #imm_data, @SP+, @-SPのとき

【0115】12-2. 比較、テスト命令

〔ニモニック〕

30 CMP src1, src2

〔命令の機能〕

src2-src1, flags affected

比較、符号拡張と比較

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図66に示す。

〔フラッグ変化〕図67に示す。

〔解説〕src1オペランドをsrc2オペランドと比較し、その結果によりPSB(L_flag, Z_flag)をセットする。src1オペランドのサイズとsrc2オペランドのサイズが異なる時は、サイズの小さい方のオペランドが符号拡張された上で比較される。なお、EaR!I, ShR!Iのモードではイミディエートを禁止しているが、@SP+は可能である。CMP @SP+, @SP+の場合、スタックポインタはオペランドサイズの2倍だけ変化し、他の命令と比較すると変則的であるが、この命令はスタックマシンをシミュレートする場合に使用することがある。

CMP: Zはいわゆるtest命令であるが、動作やフラッグ変化が同じであるため、CMPの短縮形の一つと

して扱っている。

以下では、

$src1 = [S0, S1, \dots, Ss-2, Ss-1]$

$src2 = [D0, D1, \dots, Dd-2, Dd-1]$

によってオペレーションを説明する。

($d \geq s$ の時)

$[D0, D1, \dots, Dd-s-1, Dd-s, Dd-s+1, \dots, Dd-2, Dd-1] -$

$[S0, S0, \dots, S0, S0, S1, \dots, Ss-2, Ss-1] ==>$

$d-s$ ビットだけ符号拡張

$[R0, R1, \dots, Rd-s-1, Rd-s, Rd-s+1, \dots, Rd-2, Rd-1]$ (どこにも設定されない)

($d < s$ の時)

$[D0, D0, \dots, D0, D0, D1, \dots, Dd-2, Dd-1] -$

$s-d$ ビットだけ符号拡張

$[S0, S1, \dots, Ss-d-1, Ss-d, Ss-d+1, \dots, Ss-2, Ss-1] ==>$

$[F0, F1, \dots, Fs-d-1, Fs-d, Fs-d+1, \dots, Fs-2, Fs-1]$ (どこにも設定されない)

$L_flag \star S[D] < S[S]$

SUB命令と同じ

$Z_flag [R0 \sim d-1] = 0$

($d \geq s$ の時)

$\star [F0 \sim s-1] = 0$

($d < s$ の時)

【0116】 [プログラム例外]

・予約命令例外

・RR='11' のとき

・SS='11' のとき

・EaR, ShR が@-SPのとき

・EaR!I, ShR!I が#imm_data, @-SP のとき

[ニモニック]

CMPU $src1, src2$

[命令の機能]

$src2 - src1, flags\ affected$

ゼロ拡張と比較

[命令オプション] なし

[命令ビットパターンとアセンブラ表記] 図68に示す。

[フラグ変化] 図69に示す。

[解説] $src1$ オペランドを $src2$ オペランドと比較し、その結果によりPSB (L_flag, Z_flag) をセットする。 $src1$ オペランドのサイズと $src2$ オペランドのサイズが異なる時は、サイズの小さい方のオペランドがゼロ拡張された上で比較される。EaR!Iのモードではイミディエートを禁止しているが、@SP+は可能である。

71
以下では、

src1 = {S0, S1, ..., Ss-2, Ss-1}
src2 = {D0, D1, ..., Dd-2, Dd-1}

によってオペレーションを説明する。

(d ≥ s の時)

{D0, D1, ..., Dd-s-1, Dd-s, Dd-s+1, ..., Dd-2, Dd-1} -
[0, 0, ..., 0, S0, S1, ..., Ss-2, Ss-1] ==>

d-s ビットだけゼロ拡張

{R0, R1, ..., Rd-s-1, Rd-s, Rd-s+1, ..., Rd-2, Rd-1} (どこにも設定されな
い)

(d < s の時)

[0, 0, ..., 0, D0, D1, ..., Dd-2, Dd-1] -

s-d ビットだけゼロ拡張

{S0, S1, ..., Ss-d-1, Ss-d, Ss-d+1, ..., Ss-2, Ss-1} ==>

{F0, F1, ..., Fs-d-1, Fs-d, Fs-d+1, ..., Fs-2, Fs-1} (どこにも設定されな
い)

L_flag ☆ U[D] < U[S]

SUBU 命令と同じ

Z_flag [R0~d-1] = 0

(d ≥ s の時)

☆ [F0~s-1] = 0

(d < s の時)

【0117】【プログラム例外】

・予約命令例外

・RR='11' のとき

・SS='11' のとき

・EaR が@-SPのとき

・EaR!l が#imm_data.@-SP のとき

〔ニモニック〕

CHK bound, index, xreg

〔命令の機能〕

check upper and lower bounds

配列の範囲のチェック

〔命令オプション〕

/S 下限値を引く

/N 下限値を引かない (デフォルト)

〔命令ビットパターンとアセンブラ表記〕図70に示
す。

〔フラッグ変化〕図71に示す。

〔解説〕配列のインデクスの範囲のチェックとレジスタ
へのロードを行なう。boundの指すアドレスには上
限值と下限値が組みになって置かれており、その上限
値、下限値とindexによりフェッチされた比較値オ
ペランドが比較される。boundの実効アドレスに置
かれているのが上限値であり、(boundの実効アド
レス+オペランドサイズ)のアドレスに置かれているの
が下限値である。比較は符号付き整数として行なわれ
る。比較値が上限値と下限値の間に入っていない場合
には、V_flagがセットされるので、続けてTRAP
命令を実行することにより、例外処理を起動すること
ができる。/Sを指定した場合、比較値から下限値を引
いたものがレジスタxregにロードされる。/Sを指定
しない場合、比較値はそのままレジスタxregにロー
ドされる。比較値をレジスタにロードするのは、次にそ
れを配列のインデクスのアドレス計算に使うことが多い
ためである。

50 オペレーション:

73

74

```

    tmp = mem[address_of__bound + operand_size]
    if (index ≥ mem[address_of__bound].or. index < tmp)
    then
        set V_flag:
        if (A == 1)
        then
            index - tmp ==> xreg
        else
            index ==> xreg

```

ただし、'address_of__' は'mem[...]' の逆演算子であり、

bound とmem [address_of__bound]が同じ意味になる。

下限の方は、値が一致した場合に範囲内と見なされるが、上限の方は、値が一致した場合には範囲外と見なされる。例えば bound のメモリが(0,100) となっていると、CHK で範囲内となるのはindex が0～99の場合である。

L_flag, Z_flagは、下限値とindex との比較結果にしたがってCMPと同様にセットされるが、L_flag=1となるのは、

index<下限値

の場合である。つまり、図72のようになる。

①上限値<下限値の場合には、下限値との比較により1になることもある。この場合、index-下限値の演算結果によってフラグがセットされることになる。次の3つの命令は、すべて第二オペランドが第一オペランド (CHKでは第一オペランドboundの下限値) よりも小さい時にL_flagがセットされるという仕様である。

```

CMP    src1,src2
SUB    src,dest
CHK    bound,index,xreg

```

CHK命令では、上限値≥下限値のチェックは特に行なわない。上限値と下限値の大小にかかわらず、「オペレーション」に書かれたものと等価の動作を行なうものとする。EaRdRで指定されるアドレッシングモードでは、レジスタ直接Rn, @-SP, @SP+, #imm_dataのモードは使用できない。どうしてもレジスタ上の値と比較したい場合には、CHKではなくCMPを2回行なえばよい。

【0118】(プログラム例外)

・予約命令例外

- ・RR='11' のとき
- ・EaR が@-SPのとき
- ・EaRdR がRn, #imm_data, @SP+, @-SPのとき

【0119】12-3. 算術演算命令

〔ニモニック〕

ADD src, dest

〔命令の機能〕

dest+src==> dest

加算、符号拡張と加算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図73に示

30 す。

〔フラグ変化〕図74に示す。

〔解説〕ソースオペランドをデスティネーションオペランドに加算する。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースが符号拡張された上で加算される。デスティネーションのサイズが小さく、演算結果がデスティネーションのサイズの符号付き整数として表現できない時は、V_flagがセットされる。なお、L_formatのADD:L @SP+, SPについては、ADD:G (initSP+4)+@initSP==>SP

40

の動作を行なうのが望ましい。しかし、インプリメント上、L_formatではこのような動作を行なうのが難しい場合があるので、ADD:L @SP+, SPの動作についてはインプリメント依存とする。これは、SUB:L, CMP:L, INDEXも同様である。詳しくは付録12を参照のこと。

75

76

(d ≥ s の時)

$[D0, D1, \dots, Dd-s-1, Dd-s, Dd-s+1, \dots, Dd-2, Dd-1] +$
 $[S0, S0, \dots, S0, S0, S1, \dots, Ss-2, Ss-1] ==>$

d-s ビットだけ符号拡張

 $[R0, R1, \dots, Rd-s-1, Rd-s, Rd-s+1, \dots, Rd-2, Rd-1]$ (destに設定される)

(d < s の時)

 $[D0, D0, \dots, D0, D0, D1, \dots, Dd-2, Dd-1] +$

s-dビットだけ符号拡張

 $[S0, S1, \dots, Ss-d-1, Ss-d, Ss-d+1, \dots, Ss-2, Ss-1] ==>$
 $[F0, F1, \dots, Fs-d-1, Fs-d, Fs-d+1, \dots, Fs-2, Fs-1] ==>$
 $[R0, R1, \dots, Rd-2, Rd-1]$ (destに設定される)

F0, F1, ..., Fs-d-1

s-dビットがカットされる

L_flag☆ $S[D] + S[S] < 0$

結果が負になることを表わす。

(M_flagも結果の正負を表わすが、M_flagが正しい正負
 を表示するのはオーバーフローのない時に限られる。)

M_flag R0

Z_flag $[R0 \sim d-1] = 0$ V_flag $S[D] + S[S] < -2^{(d-1)} \text{ .or. } S[D] + S[S] \geq +2^{(d-1)}$ X_flag☆ いずれの場合も、destのサイズからの桁上げが X_flagにセ
ットされる。

(d ≥ s の時)

 $U[D0, D1, \dots, Dd-s-1, Dd-s, Dd-s+1, \dots, Dd-2, Dd-1] +$
 $U[S0, S0, \dots, S0, S0,$
 $S1, \dots, Ss-2, Ss-1] \geq +2^d$

d-s ビットだけ符号拡張

(d < s の時)

 $U[D0, D1, \dots, Dd-2, Dd-1] +$
 $U[Ss-d, Ss-d+1, \dots, Ss-2, Ss-1]$
 $\geq +2^d$

S0, S1, ..., Ss-d-1

s-d ビットがカットされる

【0120】 (プログラム例外)

・予約命令例外

・RR='11' のとき

・MM='11' のとき

・EaR, ShRwが@-SPのとき

・EaM, ShM が\$imm_data, @SP+, @-SPのとき

[ニモニック]

ADDU src, dest

[命令の機能]

dest + src ==> dest

ゼロ拡張と加算

[命令オプション] なし

50 [命令ビットパターンとアセンブラ表記] 図75に示

す。

〔フラッグ変化〕図76に示す。

〔解説〕ソースオペランドをデスティネーションオペランドに加算する。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースがゼロ拡張された上で加算される。デスティネーション*

($d \geq s$ の時)

$[D0, D1, \dots, Dd-s-1, Dd-s, Dd-s+1, \dots, Dd-2, Dd-1] +$

$[0, 0, \dots, 0, S0, S1, \dots, Ss-2, Ss-1] ==>$

$d-s$ ビットだけゼロ拡張

$[R0, R1, \dots, Rd-s-1, Rd-s, Rd-s+1, \dots, Rd-2, Rd-1]$ (destに設定される)

($d < s$ の時)

$[0, 0, \dots, 0, D0, D1, \dots, Dd-2, Dd-1] +$

$s-d$ ビットだけゼロ拡張

$[S0, S1, \dots, Ss-d-1, Ss-d, Ss-d+1, \dots, Ss-2, Ss-1] ==>$

$[F0, F1, \dots, Fs-d-1, Fs-d, Fs-d+1, \dots, Fs-2, Fs-1] ==>$

$[R0, R1, \dots, Rd-2, Rd-1]$ (destに設定される)

$F0, F1, \dots, Fs-d-1$

$s-d$ ビットがカットされる。

$L_flag = 0$

$M_flag = R0$

$Z_flag = [R0 \sim d-1] = 0$

$V_flag = U[D] + U[S] \geq +2^d$

X_flag ☆ いずれの場合も、destのサイズからの桁上げが X_flag にセットされる。

($d \geq s$ の時)

$U[D0, D1, \dots, Dd-s-1, Dd-s, Dd-s+1, \dots, Dd-2, Dd-1] +$

$U[0, 0, \dots, 0, S0, S1, \dots, Ss-2, Ss-1] \geq +2^d$

$d-s$ ビットだけゼロ拡張

ADDU命令の V_flag と同じ

($d < s$ の時)

$[D0, D1, \dots, Dd-2, Dd-1] +$

$U[Ss-d, Ss-d+1, \dots, Ss-2, Ss-1]$

$\geq +2^d$

$S0, S1, \dots, Ss-d-1$

$s-d$ ビットがカットされる

【0121】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・MM='11' のとき

79

- ・EaR が@-SPのとき
- ・EaM が#imm_data, @SP+, @-SPのとき

〔ニモニック〕

ADDX src, dest

〔命令の機能〕

dest ← src + X_flag → dest

キャリーを含めた加算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図77に示す。

〔フラッグ変化〕図78に示す。

ADD @src.W, @dest1.W

ADDX #0, @dest2.W

のADDX:E #0 といった形で利用することがある。

(d ≥ s の時)

[D0.D1....Dd-s-1.Dd-s.Dd-s+1....Dd-2.Dd-1] +

[S0.S1....Ss-1.Ss.Ss+1....Ss-2.Ss-1] + X_flag ==>

d-s ビットだけ符号拡張

[R0.R1....Rd-s-1.Rd-s.Rd-s+1....Rd-2.Rd-1] (destに設定される)

(d < s の時)

{ D0.D0....D0.D0.D1....Dd-2.Dd-1} +

s-d ビットだけ符号拡張

[S0.S1....Ss-d-1.Ss-d.Ss-d+1....Ss-2.Ss-1] + X_flag ==>

[F0.F1....Fs-d-1.Fs-d.Fs-d+1....Fs-2.Fs-1] ==>

[R0.R1....Rd-2.Rd-1] (destに設定される)

F0.F1....Fs-d-1

s-d ビットがカットされる。

L_flag☆ S[D] + S[S] + X_flag < 0

符号付きの数と見て演算を行ない、結果が負になることを表わす。d ≠ s の場合には、オペランドが符号拡張してから比較される。(M_flagも結果の正負を表わすが、 M_flagが正しい正負を表示するのはオーバーフローのない時に限られる。)

M_flag R0

Z_flag [R0~d-1] = 0 .and. previous Z_flag

V_flag S[D] + S[S] + X_flag < -2^(d-1) .or.

80

〔解説〕ソースオペランドとキャリーをデスティネーションオペランドに加算する。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースが符号拡張された上で加算される。Z_flagでは、フラッグ値を累積できるようになっている。また、ADDXとADDのフラッグ変化は、符号拡張／ゼロ拡張を含めてほとんど同じである。ADDとADDXでフラッグ変化の異なるのは、Z_flagのみである。なお、ADDX, SUBXの累種サイズ間演算については、例えば8バイトの数dest2~dest1に4バイトの数srcを加える場合、

81

82

$$S[D] + S[S] + X_flag \geq +2^{(d-1)}$$

符号付きの数と見て演算を行ない、結果がオーバーフローすることを表わす。d ≠ s の場合にはオペランドが符号拡張される。

X_flag ☆ いずれの場合も、destのサイズからの桁上げが X_flagにセットされる。

(d ≥ s の時)

$$\begin{aligned} & U[D0.D1....Dd-s-1.Dd-s.Dd-s+1....Dd-2.Dd-1] + \\ & U[S0.S0.....S0.S0.S1....Ss-2.Ss-1] + X_flag \\ & \geq +2^d \end{aligned}$$

d-s ビットだけ符号拡張

d > s の場合には符号拡張を行なう。これは、destや他のフラッグの設定の処理と共通化したためである。しかし、符号拡張後の演算、比較ではオペランドが符号なしの数として扱われる。

(d < s の時)

$$\begin{aligned} & U[D0.D1....Dd-2.Dd-1] + \\ & \quad U[Ss-d.Ss-d+1....Ss-2.Ss-1] + X_flag \\ & \geq +2^d \end{aligned}$$

$$S0.S1....Ss-d-1$$

s-d ビットがカットされる

【0122】 (プログラム例外)

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・BaR が@-SPのとき
- ・BaM がYimm_data.@SP+, @-SPのとき

〔ニモニック〕

SUB src, dest

〔命令の機能〕

dest - src ==> dest

減算、符号拡張と減算

〔命令オプション〕 なし

〔命令ビットパターンとアセンブラ表記〕 図79に示 40

す。

〔フラッグ変化〕 図80に示す。

〔解説〕 ソースオペランドをデスティネーションオペランドから減ずる。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースが符号拡張された上で減算される。デスティネーションのサイズが小さく、演算結果がデスティネーションのサイズの符号付き整数として表現できない時は、V_flagがセットされる。

(d ≥ s の時)

$$\begin{aligned} & [D0.D1....Dd-s-1.Dd-s.Dd-s+1....Dd-2.Dd-1] - \\ & [S0.S0.....S0.S0.S1....Ss-2.Ss-1] ==> \end{aligned}$$

83

d-sビットだけ符号拡張

[R0, R1, ..., Rd-s-1, Rd-s, Rd-s+1, ..., Rd-2, Rd-1] (destに設定される)

(d < s の時)

{D0, D0, ..., D0, D0, D1, ..., Dd-2, Dd-1} -

s-dビットだけ符号拡張

{S0, S1, ..., Ss-d-1, Ss-d, Ss-d+1, ..., Ss-2, Ss-1} ==>

{F0, F1, ..., Fs-d-1, Fs-d, Fs-d+1, ..., Fs-2, Fs-1} ==>

[R0, R1, ..., Rd-2, Rd-1] (destに設定される)

F0, F1, ..., Fs-d-1

s-dビットがカットされる。

L_flag☆ S[D] - S[S] < 0

結果が負になることを表わす。

(M_flagも結果の正負を表わすが、 M_flagが正しい

正負を表示するのはオーバーフローのない時に限られる

。)

M_flag R0

Z_flag [R0 ~ d-1] = 0

V_flag S[D] - S[S] < -2^(d-1) .or.S[D] - S[S] ≥ +2^(d-1)

X_flag☆ いずれの場合も、destのサイズからの桁下げが X_flag

にセットされる。

(d ≥ s の時)

U{D0, D1, ..., Dd-s-1, Dd-s, Dd-s+1, ..., Dd-2, Dd-1} -

U{S0, S0, ..., S0, S0, S1, ..., Ss-2, Ss-1} < 0

d-sビットだけ符号拡張

(d < s の時)

U{ D0, D1, ..., Dd-2, Dd-1} -

U{Ss-d, Ss-d+1, ..., Ss-2, Ss-1} < 0

S0, S1, ..., Ss-d-1

s-dビットがカットされる

dest - src ==> dest

ゼロ拡張と減算

[命令オプション] なし

【0123】 [プログラム例外]

・予約命令例外

・RR='11' のとき

・MM='11' のとき

・EaR, ShRWがθ-SPのとき

・EaM, ShM が#imm_data, θSP+, θ-SPのとき

[ニモニック]

SUBU src, dest

[命令の機能]

40 [命令ビットパターンとアセンブラ表記] 図81に示す。

[フラッグ変化] 図82に示す。

[解説] ソースオペランドをデスティネーションオペランドから減ずる。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースがゼロ拡張された上で減算される。デスティネーションのサイズが小さく、演算結果がデスティネーションのサイズの符号なし整数として表現できない時は、V_flagがセットされる。

85

86

(d ≥ s の時)

[D0, D1, ..., Dd-s-1, Dd-s, Dd-s+1, ..., Dd-2, Dd-1] -

{ 0, 0, ..., 0, S0, S1, ..., Ss-2, Ss-1 } ==>

d-s ビットだけゼロ拡張

[R0, R1, ..., Rd-s-1, Rd-s, Rd-s+1, ..., Rd-2, Rd-1] (destに設定される)

(d < s の時)

[0, 0, ..., 0, D0, D1, ..., Dd-2, Dd-1] -

s-d ビットだけゼロ拡張

[S0, S1, ..., Ss-d-1, Ss-d, Ss-d+1, ..., Ss-2, Ss-1] ==>

[F0, F1, ..., Fs-d-1, Fs-d, Fs-d+1, ..., Fs-2, Fs-1] ==>

[R0, R1, ..., Rd-2, Rd-1] (destに設定される)

F0, F1, ..., Fs-d-1

s-dビットがカットされる。

L_flag ☆ U[D] - U[S] < 0

結果が負になることを表わす。

(M_flagも結果の正負を表わすが、 M_flagが正しい
正負を表示するのはオーバーフローのない時に限られる
。)

M_flag R0

Z_flag [R0 ~ d-1] = 0

V_flag U[D] - U[S] < 0

SUBU命令の L_flagと同じ

X_flag ☆ いずれの場合も、destのサイズからの桁下げが X_flag
にセットされる。

(d ≥ s の時)

U[D0, D1, ..., Dd-s-1, Dd-s, Dd-s+1, ..., Dd-2, Dd-1] -

U[0, 0, ..., 0, S0, S1, ..., Ss-2, Ss-1] < 0

d-sビットだけゼロ拡張

SUB命令の X_flag、SUBU命令の L_flag、V_flagと同じ

(d < s の時)

U[D0, D1, ..., Dd-2, Dd-1] -

U[Ss-d, Ss-d+1, ..., Ss-2, Ss-1] < 0

S0, S1, ..., Ss-d-1

s-dビットがカットされる

【0124】 [プログラム例外]

・予約命令例外

・RR='11' のとき

・MM='11' のとき

・EaR が@-SPのとき

・BaM がimm_data, @SP+, @-SPのとき

87

〔モニニック〕

SUBX src, dest

〔命令の機能〕

dest - src - X_flag ==> dest

キャリーを含めた減算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図83に示す。

〔フラッグ変化〕図84に示す。

(d ≥ s の時)

[D0.D1....Dd-s-1.Dd-s.Dd-s+1....Dd-2.Dd-1] -

[S0.S0.....S0. S0. S1....Ss-2.Ss-1] - X_flag ==>

d-s ビットだけ符号拡張

[R0.R1....Rd-s-1.Rd-s.Rd-s+1....Rd-2.Rd-1] (destに設定される)

(d < s の時)

[D0.D0.....D0. D0. D1....Dd-2.Dd-1] -

s-dビットだけ符号拡張

[S0.S1....Ss-d-1.Ss-d.Ss-d+1....Ss-2.Ss-1] - X_flag ==>

88

〔解説〕ソースオペランドとキャリーをデスティネーションオペランドから減ずる。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースが符号拡張された上で減算される。Z_flagではフラッグ値を累積できるようになっている。また、SUBXとSUBのフラッグ変化は、符号拡張／ゼロ拡張を含めてほとんど同じである。SUBとSUBXでフラッグ変化の異なるのは、Z_flagのみである。

89
 [R0, R1, ..., Rd-2, Rd-1] (destに設定される)

90

R0, R1, ..., Rd-1

s-dビットがカットされる。

L_flag ☆ $S[D] - S[S] - X_flag < 0$

符号付きの数と見て演算を行ない、結果が負になることを表わす。d ≠ s の場合にはオペランドが符号拡張されてから比較される。

(M_flagも結果の正負を表わすが、M_flagが正しい正負を表示するのはオーバーフローのない時に限られる。)

M_flag R0

Z_flag $[R0 \sim d-1] = 0$.and. previous Z_flag

V_flag $S[D] - S[S] - X_flag < -2^{(d-1)}$.or.
 $S[D] - S[S] - X_flag \geq +2^{(d-1)}$

符号付きの数と見て演算を行ない、結果がオーバーフローすることを表わす。d ≠ s の場合にはオペランドが符号拡張される。

X_flag ☆ いずれの場合も、destのサイズからの桁下げが X_flag にセットされる。

(d ≥ s の時)

$U[D0, D1, \dots, Dd-s-1, Dd-s, Dd-s+1, \dots, Dd-2, Dd-1] -$

$U[S0, S0, \dots, S0, S0, S1, \dots, Ss-2, Ss-1] -$

$X_flag < 0$

d-s ビットだけ符号拡張

d > s の場合には符号拡張を行なう。これは、destや他のフラグの設定の処理と共通化したためである。しかし、符号拡張後の演算、比較ではオペランドが符号なしの数として扱われる。

(d < s の時)

$U[D0, D1, \dots, Dd-2, Dd-1] -$

$U[Ss-d, Ss-d+1, \dots, Ss-2, Ss-1] -$

$X_flag < 0$

S0, S1, ..., Ss-d-1

s-dビットがカットされる

【0125】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・MM='11' のとき

・BaR が@-SPのとき

・BaM が#imm_data, @SP+, @-SPのとき

〔ニモニック〕

MUL src, dest

〔命令の機能〕

40 dest * src ==> dest

乗算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図85に示す。

〔フラグ変化〕図86に示す。

〔解説〕ソースオペランドをデスティネーションオペランドに乗ずる。乗算は符号付きで行われ、オペランドも符号付き整数とみなされる。この命令は、被乗数のサイズと結果のサイズが等しいため、高級言語向きである。

50 デスティネーションのサイズが小さく、演算結果がデス

91

ディネーションのサイズの符号付き整数として表現できない時は、`V_flag`がセットされる。オーバーフローが生じた場合にも、`dest`にセットされるデータ（正しい結果の下位ビット）が基準となって`M_flag`、`Z_flag`がセットされる。例えば、`R0=H'10000`で

```
MUL.W  #H'10000,R0
```

を実行した場合、積が`H'100000000`となるため、`R0=0`（下位ビット）、`V_flag=1`、`Z_flag=1`となる。

【0126】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・BaR が@-SPのとき
- ・BaM が#imm_data.@SP+,@-SPのとき

〔ニモニック〕

```
MULU    src, dest
```

〔命令の機能〕

```
dest * src ==> dest
```

符号なし乗算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図87に示す。

〔フラッグ変化〕図88に示す。

〔解説〕ソースオペランドをデスティネーションオペランドに乗ずる。乗算は符号なしで行われ、オペランドも*
[MULXのオペレーション]

```
dest[0:31] × src[0:31] ==> tmp1[0:63]
```

```
tmp1[32:63] ==> tmp[0:31]
```

```
tmp1[0:31] ==> dest[0:31]
```

MULXでは、得るべき結果が`dest`、`tmp`の二つあるので、両者が重なった場合（`dest`で`tmp`と同じレジスタを指定した場合）の処置が問題となる。一般に、`tmp`（MULXの上位桁）の方は次の桁への桁上がりとして用いられることが多いので、最終桁の計算などでは使用しないこともある。したがって、両者が重なった場合には、`dest`に設定すべき値（MULXの下位桁）の方が残るものとする。（付録12参照）

MULXの`M_flag`、`Z_flag`のフラッグ変化は、`dest`を基準とする。`tmp`に設定される値は、これらのフラッグには影響しない。このような仕様になったのは、次のような理由による。・ADDX、SUBXなどのフラッグ変化の仕様に合わせたため。（ADDX、SUBXでは、`X_flag`がセットされていても、`dest`が0であれば`Z_flag`がセットされる。）

92

*符号なし整数とみなされる。デスティネーションのサイズが小さく、演算結果がデスティネーションのサイズの符号なし整数として表現できない時は、`V_flag`がセットされる。

【0127】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・BaR が@-SPのとき
- ・BaM が#imm_data.@SP+,@-SPのとき

〔ニモニック〕

```
MULX    src, dest, tmp
```

〔命令の機能〕

```
dest * src ==> reg&dest (double size)
```

拡張乗算、サイズが大きくなる

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図89に示す。

20 〔フラッグ変化〕図90に示す。

〔解説〕ソースオペランドをデスティネーションオペランドに乗ずる。この命令は、積が倍長で得られるため、`src`と`dest`のほかに、積の上位ビットを入れるためのテンポラリレジスタ`tmp`を指定する。サイズは32ビットに固定（32/64から選択）とする。乗算は符号なしで行われ、積のサイズは被乗数のサイズの2倍になる。

・多倍長演算を考えた場合には、`tmp&dest`のみでフラッグを変化させてもあまり意味がない。フラッグ本来の意味で変化させるためには、全体の値を通して判定することが必要であり、個々の命令だけでは対処できない。`tmp&dest`でフラッグ変化を行なったとしても、結局中途半端である。

例：

〔実行前〕

```
R1=H'00000000 dest=H'20000000 src=H'40000000
```

```
MULX @src,@dest,R1
```

〔実行後〕

【0128】

〔数6〕

50

93

tmp=H' 0800000000000000

```

    .....
    R1      dest

```

dest に設定される値が0なので、
Z_flagがセットされる。

【0129】なお、MULX, DIVXでは、ADD
X, SUBXとは異なり、Z_flagは累積した変化*
・予約命令例外

・|R|=11' のとき

注) !=0 のときは予約命令例外としては検出しない。

・EaR が@-SPのとき

・EaMRが#imm_data, @SP+, @-SPのとき

〔ニモニック〕

DIV src, dest

〔命令の機能〕

dest/src==> dest

除算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図91に示す。

〔フラッグ変化〕図92に示す。

〔解説〕ソースオペランドでデスティネーションオペランドを割る。除算は符号付きで行われ、オペランドも符号付き整数とみなされる。この命令は、被除数のサイズと結果のサイズが等しいため、高級言語向きである。商は0方向に丸められ、余りの符号は被除数と同じになる。

例:

```

10/3      →  商=3,      剰余=1
(-10)/3   →  商=(-3),   剰余=(-1)
10/(-3)   →  商=(-3),   剰余=1

```

src=0の場合には、ゼロ除算例外(ZDE)となる。ゼロ除算の場合、V_flagがセットされて例外例:

src=H' ffff=(-1), dest=H' 8000=(-32768)

でDIV.H を実行した場合

=> dest=H' 8000, V_flag=1

destのH' 8000は、正しい結果

(H' ... 008000=32768) の下位ビットと考えることもできるし

、destが変化しなかったと考えることもできる。

【0131】〔プログラム例外〕

94

*をするわけではない。F_flagによってtmp=0のテストが可能である。!=0の場合は、動作を保証しない。実際「本発明装置」では!=0の場合、srcサイズを、!R(8ビットまたは16ビット)としてオペランドのフェッチを行ない、それを32ビットに符号拡張して命令が実行される。ただし、dest, tmpは!Rによらず常に32ビットとして扱われる。

【0130】〔プログラム例外〕

処理が起動されるが、destの値は変化しない。このとき、destのライトアクセスを行なうかどうか、すなわち、同じ値を書き込むか、何も書き込まないかは規定しないものとする。また、V_flag以外のフラッグも変化しない。これは、destに合わせたため、および、例外処理プログラムで例外発生要因を解析するためには、できるだけ以前の状態(フラッグを含めて)が保存されている方が望ましいためである。DIVで0除算以外の場合にオーバーフローが発生するのは(最小負数)÷(-1)の場合のみである。DIVはDIVXとは異なり、コンパイラの生成する普通の演算命令であるため、できるだけ他の演算命令と同じ仕様にする方が望ましい。そこで、この場合のフラッグの変化は、それぞれのフラッグの意味を生かして、

V_flag=1, L_flag=0, M_flag=1, Z_flag=0

〔最小負数÷(-1)の時〕

とする。オーバーフローするのは(最小負数÷(-1))に限るので、正しい結果の下位ビットがdestにセットされたと考えても、結局 destは変化しない。正しい結果の下位ビットになったと考えると結局同じ値である。

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・EaR が@-SPのとき
- ・EaM が#imm_data, @SP+, @-SPのとき

・ゼロ除算例外

- ・src=0のとき

〔ニモニック〕

DIVU src, dest

〔命令の機能〕

dest/src==> dest

符号なし除算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図93に示す。

〔フラッグ変化〕図94に示す。

〔解説〕ソースオペランドでデスティネーションオペランドを割る。除算は符号なしで行われ、オペランドも符号なし整数とみなされる。src=0の場合には、ゼロ除算例外(ZDE)となる。ゼロ除算の場合、V_flagがセットされて例外処理が起動されるが、destは変化しない。このとき、destのライトアクセスを行なうかどうか、すなわち、同じ値を書き込むか、何も書き込まないかは規定しないものとする。また、V_flag以外のフラッグも変化しない。これは、destに合わせたため、および、例外処理プログラムで例外発生要因を解析するためには、できるだけ以前の状態(フラッグを含めて)が保存されている方が望ましいためである。DIVU命令では、0除算以外の場合に、オーバーフローが発生してV_flagがセットされることはない。したがって、0除算以外の場合は必ずV_flagがクリアされる。

【0132】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・EaR が@-SPのとき
- ・EaM が#imm_data, @SP+, @-SPのとき

・ゼロ除算例外

- ・src=0のとき

〔ニモニック〕

DIVX src, dest, tmp

〔命令の機能〕

reg<dest / src ==> dest.reg (quotient, remainder)

拡張除算、サイズが小さくなる、剰余を出す

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図95に示す。

〔フラッグ変化〕図96に示す。

〔解説〕ソースオペランドでデスティネーションオペランドを割る。この命令は、多倍長除算のプリミティブとなるため、srcとdestのほかに、拡張演算のためのテンポラリ値(剰余)を置くレジスタを指定する。サイズは32ビットに固定(32/64から選択)とする。除算は符号なしで行われ、被除数のサイズは除数のサイズの2倍になる。

〔DIVXのオペレーション〕

```
10 concatenate(tmp[0:31],dest[0:31]) ==> tmp[0:63]
   quo(tmp[0:63],src[0:31]) ==> dest[0:31]
   rem(tmp[0:63],src[0:31]) ==> tmp [0:31]
```

DIVXでは、得るべき結果がdest, tmpの二つあるので、両者が重なった場合(destでtmpと同じレジスタを指定した場合)の処置が問題となる。一般に、tmp(DIVXの剰余)の方は次の桁への桁下がりとして用いられることが多いので、最終桁の計算などでは使用しないこともある。したがって、両者が重なった場合には、destに設定すべき値(DIVXの商)の方が残るものとする。なお、DIVXは被除数が多倍長の場合に使用できる命令であるが、除数が多倍長になった場合には、DIVXが使用できず、プログラムによってシフトや減算を繰り返しながら割り算を進めなければならない。この際、多倍長のシフト演算が必要になる。多倍長のシフトを実現するため、X_flagを通したローテイト命令(SHXR, SHXL)が用意されている。DIVXのM_flag, Z_flagのフラッグ変化は、dest(商)を基準とする。tmpに設定される値(剰余)は、これらのフラッグには影響しない。ただし、F_flagによってtmp=0のテストが可能である。MULX, DIVXではADDX, SUBXとは異なり、Z_flagは累積した変化をするわけではない。DIVXで結0がオーバーフローした場合、MOV, ADD, SUB, MULでのオーバーフローと仕様を合わせるという意味では、正しい結果の下位ビットがdestに設定されるのが望ましい。しかし、ADD, SUBのように、オーバーフローの時も自然に正しい結果の下位ビットが得られるものと違って、除算の場合には上位ビットから計算を行なうため、アルゴリズムの関係で正しい結果の下位ビットを得るのが難しい。したがって、DIVXのオーバーフローの場合には、destを変化させないという仕様にする。DIVXで商がdestに入らず、オーバーフローが発生した場合には、V_flag以外のフラッグは変化しない。これは、DIVXでオーバーフローが発生した場合に、destが変化しないことに合わせたものである。src=0の場合には、ゼロ除算例外(ZDE)となる。ゼロ除算の場合dest, tmpは変化しない。このとき、destのライトアクセスを行なうかどうか、すな

97

わち、同じ値を書き込むか、何も書き込まないかは規定しないものとする。また、V__flag以外のフラッグも変化しない。これは、destに合わせたため、および、例外処理プログラムで例外発生要因を解析するためには、できるだけ以前の状態（フラッグを含めて）が保存されている方が望ましいためである。!=0の場合は、動作を保証しない。実際「本発明装置」では!=0*

・予約命令例外

・RR='11' のとき

注) !=0 のときは予約命令例外としては検出しない。

・BaR が@-SPのとき

・BaMRが#imm_data, @SP+, @-SPのとき

・ゼロ除算例外

・src=0のとき

〔ニモニック〕

REM src, dest

〔命令の機能〕

dest % src ==> dest

剰余

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図97に示す。

〔フラッグ変化〕図98に示す。

〔解説〕ソースオペランドでデスティネーションオペランドを割り、その剰余を求める。除算は符号付きで行われ、オペランドも符号付き整数とみなされる。この命令は、被除数のサイズと剰余のサイズが等しいため、高級言語向きである。商は0方向に丸められ、余りの符号は被除数と同じになる。

例：

10/3	→	商=3,	剰余=1
(-10)/3	→	商=(-3),	剰余=(-1)
10/(-3)	→	商=(-3),	剰余=1

src=0の場合には、ゼロ除算例外(ZDE)となる。ただし、REMで0除算を行なった場合には、オーバーフローをクリアして例外処理を起動するようにする。REM命令では、DIV命令とは異なり、0除算をしてもdest(剰余)がオーバーフローするわけではないので、V__flagはクリアしておく方が合理的である。また、V__flagをクリアしておくと、例外処理の中でDIVによるエラーかREMによるエラーかが判定しやすい。0除算の場合、destは無変化である。destに対してメモリアクセスを行なうか(readまたは同じ値でread-modify-write)、アクセスを行なわないか、については、インプリメント方法を縛ることになるので、規定しない。

【0134】〔プログラム例外〕

98

*の場合、srcサイズを、!R(8ビットまたは16ビット)としてオペランドのフェッチを行ない、それを32ビットに符号拡張して命令が実行される。ただし、dest, tmpは!Rによらず常に32ビットとして扱われる。

【0133】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・MM='11' のとき

・BaR が@-SPのとき

・BaM が#imm_data, @SP+, @-SPのとき

20 ・ゼロ除算例外

・src=0のとき

〔ニモニック〕

REMU src, dest

〔命令の機能〕

dest % src ==> dest

符号なし除算による剰余

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図99に示す。

30 〔フラッグ変化〕図100に示す。

〔解説〕ソースオペランドでデスティネーションオペランドを割り、その剰余を求める。除算は符号なしで行われ、オペランドも符号なし整数とみなされる。srcとdestのサイズの違う場合にはゼロ拡張が行なわれる。この命令は、被除数のサイズと剰余のサイズが等しいため、高級言語向きである。src=0の場合には、ゼロ除算例外(ZDE)となる。0除算の場合の処置はREMと同様である。

【0135】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・MM='11' のとき

・BaR が@-SPのとき

・BaM が#imm_data, @SP+, @-SPのとき

・ゼロ除算例外

・src=0のとき

〔ニモニック〕

NEG dest

50 〔命令の機能〕

99

100

O-dest=> dest

*す。

補数演算

〔フラッグ変化〕図102に示す。

〔命令オプション〕なし

〔解説〕オペランドの符号を反転する。

〔命令ビットパターンとアセンブラ表記〕図101に示*

L_flag 命令実行後のdestの値が負のとき、すなわちdestの初期値が正のときセットされる。

M_flag 命令実行後のdestのMSB が1 のとき、すなわちdestの初期値が正または最小負数のときセットされる。

Z_flag 命令実行後のdestの値が0 のとき、すなわちdestの初期値が0 のときセットされる。

V_flag destの初期値が最小負数 (MSB のみ1 で他のビットはall 0) のみセットされる。

【0136】〔プログラム例外〕

・予約命令例外

・MM='11' のとき

・BaM がimm_data, @SP+, @-SPのとき

〔ニモニック〕

INDEX indexsize, subscript, xreg

〔命令の機能〕

calculate address of array

多次元配列のアドレス計算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図103に示す。

〔フラッグ変化〕図104に示す。

〔解説〕多次元配列を一次元配列に展開するためのアドレス計算として、スケールの乗算とインデクスの加算を行なう。subscriptのサイズがxregのサイズよりも小さい時は、subscriptが符号拡張される。xreg, indexsize, subscriptは符号付きの整数と考え、乗算および加算は符号付きで行なう。乗算または加算でオーバーフローを検出した場合には、V_flagをセットする。indexsizeは固定値 (immediate) で済むことが多いが、メモリ上に配列のディスクリプタを作ることを考え、汎用アドレッシングとしている。INDEX命令がCHK命令の後で実行されるならば、subscript※

・予約命令例外

・IR='11' のとき

・SS='11' のとき

注) !='0' のときは予約命令例外としては検出しない。

・BaR, EaR2が@-SPのとき

【0138】12-4. 論理演算命令

〔ニモニック〕

AND src, dest

50 〔命令の機能〕

※tはレジスタのみの指定でよい。しかし、高級言語の仕様によっては範囲をチェックしない (CHK命令を実行しない) こともあるので、メモリ上の変数をsubscriptとして使用できるように、subscriptも汎用アドレッシングとしている。

20 〔INDEXのオペレーション〕

xreg = indexsize + subscript ==> xreg

INDEX命令では、オペランドxreg, indexsize, subscriptは、すべてポインタではなく符号付きの数として扱われる。負であってもそのまま演算し、EITなどの特別な動作はしない。また、フラッグ変化 (V_flag, L_flag, M_flag, Z_flag) も一般の算術演算命令に準じたものになっている。これに関して若干補足しておく。INDEXで扱うオペランドは、ポインタというよりも配列のインデクスであり、INDEXでは配列のインデクスを一次元に展開するという処理を行なう。インデクスがポインタになるのは、付加モードのスケールリングで (×4) などを行なった後である。したがって、INDEXで扱うデータを符号付きと考えても特に不自然はないし、インデクスが負になると困るような言語では、それをチェックすることもできる。!=0のときは、動作を保証しない。実際「本発明装置」では!=0の場合、indexsizeサイズを!R (8ビットまたは16ビット) としてオペランドのフェッチを行ない、それを32ビットに符号拡張して命令が実行される。ただし、xregは!Rによらず常に32ビットとして扱われる。

【0137】〔プログラム例外〕

101

dest. and. src==> dest

論理積

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図105に示す。

〔フラッグ変化〕図106に示す。

〔解説〕ソースオペランドとデスティネーションオペランドとの論理積をとる。ソースオペランドのサイズがデスティネーションオペランドのサイズと異なる異種サイズ間の演算（AND：GのRR≠MM、AND：EのMM≠00）になった場合、命令はそのまま実行され、予約命令例外とはならないが、destに設定される結果*

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・BaR が@-SPのとき
- ・EaM が#imm_data, @SP+, @-SPのとき

〔ニモニック〕

OR src, dest

〔命令の機能〕

dest. or. src==> dest

論理和

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図107に示す。

〔フラッグ変化〕図108に示す。

〔解説〕ソースオペランドとデスティネーションオペランドとの論理和をとる。ソースオペランドのサイズがデスティネーションオペランドのサイズと異なる異種サイズ間の演算（OR：GのRR≠MM、OR：EのMM≠00）になった場合、命令はそのまま実行され、予約命令例外とはならないが、destに設定される結果は保証できない（インプリメント依存になる）ものとする。

M_flag R0
Z_flag [R0 ~d-1] = 0

【0140】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・BaR が@-SPのとき
- ・EaM が#imm_data, @SP+, @-SPのとき

〔ニモニック〕

XOR src, dest

〔命令の機能〕

dest. xor. src==> dest

排他的論理和

102

*は保証できない（インプリメント依存になる）ものとする。「本発明装置」の仕様としては、異種サイズ間の論理演算を規定していないので、この機能をソフトウェアで利用してはいけない。異種サイズ間の論理演算はあまり意味のない命令であるが、これを予約命令例外としな

M_flag R0
Z_flag [R0 ~d-1] = 0

【0139】〔プログラム例外〕

20 〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図109に示す。

〔フラッグ変化〕図110に示す。

〔解説〕ソースオペランドとデスティネーションオペランドとの排他的論理和をとる。ソースオペランドのサイズがデスティネーションオペランドのサイズと異なる異種サイズ間の演算（XOR：GのRR≠MM、XOR：EのMM≠00）になった場合、命令はそのまま実行され、予約命令例外とはならないが、destに設定される結果は保証できない（インプリメント依存になる）ものとする。

M_flag R0
Z_flag [R0 ~d-1] = 0

【0141】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・BaR が@-SPのとき
- ・EaM が#imm_data, @SP+, @-SPのとき

40

〔ニモニック〕

NOT dest

〔命令の機能〕

【0142】

〔数7〕

~dest --> dest

【0143】全ビット反転

〔命令オプション〕なし

50 〔命令ビットパターンとアセンブラ表記〕図111に示す。

す。

〔フラッグ変化〕図112に示す。

M_flag 命令実行後のdestのMSB が1 のとき、すなわちdestの初期値のMSB が0 のときセットされる。

Z_flag 命令実行後のdestの値が0 のとき、すなわちdestの初期値が0 のときセットされる。

【0144】〔プログラム例外〕

・予約命令例外

- ・MM='11' のとき
- ・EaM が#imm_data, @SP+, @-SPのとき

【0145】12-5. シフト命令

〔モニック〕

SHA count, dest

〔命令の機能〕

shift arithmetic

算術シフト

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図113に示す。

〔フラッグ変化〕図114に示す。

〔解説〕デスティネーションオペランドdestをソースオペランドcountで指定されたビット数だけ算術シフトする。一般形の命令では、countの符号によってシフト方向を指定する。countが正の時左シフト、負の時右シフトとなる。算術シフトであるため、右シフトの場合にはデスティネーションのMSB（符号ビット）が変化せず、同じ値が右側のビットにコピーされていく。左シフトの場合には、LSBに0が入り、0が左側のビットにコピーされていく。正負によるシフト方向の指定は、浮動小数点演算のエミュレーションなどに有効な場合がある。左シフトの場合はSHAの短縮形がないが、フラッグ変化がSHAと異なってもよければ、SHLの短縮形SHL: Qで代用できる。

〔左シフト(count>0)の場合〕図115に示す。

〔右シフト(count<0)の場合〕図116に示す。

なお、count=0の場合は、X_flag=0となる。SHA命令では、countのサイズとして8ビットのみが有効である。RR≠00の場合動作を保証しない。RR≠00の機能が利用できないのは、主としてインプリメント上の制約によるものである。RR≠00の場合には、「本発明装置」では、サイズRRでcountオペランドのフェッチを行ない、countの下位8ビットのみを有効としてそのまま命令を実行する。SHAは算術演算なのでL_flagをセットするが、これはdestのはじめの値の符号(MSB)を反映する。これは、L_flagが、オーバーフローやアンダーフ

*〔解説〕オペランドの各ビットの1と0を反転する。

ローが発生した場合にも、常に正しい計算結果の符号を反映するという性質を持っているためである。シフト命令の場合、オーバーフローが起こらなければ、destの符号は変化しない。右シフトの場合、または左シフトでオーバーフローがなかった場合にはL_flag=M_flagとなるが、左シフトでオーバーフローが発生した場合には、L_flagとM_flagが異なる変化をすることがある。「本発明装置」はbig-endianのチップであるため、countをビット位置の増減の意味で考えるか、2の累乗の意味で考えるかによって、シフト方向が逆になってしまう。すなわち、前者の考え方ではcount>0の時右シフトとすべきであるのに対して、後者の考え方では、little-endianの場合と同じように、count>0の時左シフトとなる。しかし、シフト命令はビット操作関係の命令よりも算術演算関係の命令に近いものであるから、countをビット位置の増減の意味で考えるよりは、2の累乗の意味で考える方が自然である。したがって、「本発明装置」ではcount>0の時左シフトという仕様になっている。SHL, SHAでは、countの絶対値が(destのサイズ+1)を越えた場合にも、指定された数だけそのままシフトを続ける。結果的に、countの絶対値が(destのサイズ+1)の場合と同じ動作になる。例えば、次のような動作をする。

```
SHA    #33, dest.W      ;dest=X_flag=0
SHL    #33, dest.W      ;dest=X_flag=0
SHA    #-33, dest.W     ;dest=X_flag= 旧destのMSB
SHL    #-33, dest.W     ;dest=X_flag=0
```

なお、X_flagを除けば、countの絶対値が(destのサイズ)に等しい場合も同じ結果になる。

【0146】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・EaR が@-SPのとき
- ・EaM, ShM が#imm_data, @SP+, @-SPのとき

〔モニック〕

SHL count, dest

〔命令の機能〕

shift logical

論理シフト

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図117に示す。

〔フラッグ変化〕図118に示す。

〔解説〕デスティネーションオペランドをソースオペランド `count` で指定されたビット数だけ論理シフトする。一般形の時、シフト方向は `count` の符号で指定する。 `count` が正の時左シフト、負の時右シフトとなる。右シフトの場合には、MSBに0が入り、0が右側のビットにコピーされていく。また、左シフトの場合には、LSBに0が入り、0が左側のビットにコピーされていく。

〔左シフト (`count > 0`) の場合〕図119に示す。

〔右シフト (`count < 0`) の場合〕図120に示す。

なお、 `count = 0` の場合は、 `X_flag = 0` となる。SHL命令では、 `count` のサイズとして8ビットのみが有効である。RR≠00の場合、動作を保証しない。RR≠00の機能が利用できないのは、主としてインプリメント上の制約によるものである。RR≠00の場合には、「本発明装置」ではサイズRRで `count` オペランドのフェッチを行ない、 `count` の下位8

【0147】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・EaR が@-SPのとき
- ・EaM, ShM が#imm_data, @SP+, @-SPのとき

〔ニモニック〕

ROT `count, dest`

〔命令の機能〕

rotate

回転

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図121に示す。

〔フラッグ変化〕図122に示す。

〔解説〕デスティネーションオペランドをソースオペランド `count` で指定されたビット数だけ回転する。すなわち、LSB (MSB) から溢れたビットをMSB (LSB) に詰めながらシフトを行なう。回転方向は `count` の符号で指定する。 `count` が正の時左回転、負の時右回転となる。回転の際、フラッグは通さない。

〔左回転 (`count > 0`) の場合〕図123に示す。

〔右回転 (`count < 0`) の場合〕図124に示す。

なお、 `count = 0` の場合は、 `X_flag = 0` となる。ROT命令では、 `count` のサイズとして8ビットのみが有効である。RR≠00の場合、動作を保証しない。RR≠00の機能が利用できないのは、主としてインプリメント上の制約によるものである。RR≠00の場合には、「本発明装置」ではサイズRRで `count` オペランドのフェッチを行ない、 `count` の下位8ビットのみを有効としてそのまま命令を実行する。ROTで `count` の絶対値が (`dest` のサイズ) を越えた場合にも、指定された数だけそのままローテイトを続ける。結果的に、 `count` を (`dest` のサイズ) で割った剰余を `count` とした場合と同じ動作になる。ただし、 `count` が (`dest` のサイズ) の整数倍 (≠0) の場合には、MSBまたはLSBに応じて `X_flag` のセットされる点が、 `count = 0` の場合とは異なる。例えば、データサイズと同じビット数だけ回転した場合、データの値そのものは変化せず、 `dest` は `count = 0` の時と同じ値になる。しかし、フラッグにはもとのデータのLSBがコピーされるため、フラッグ変化は `count = 0` の時とは異なったものになる。

【0148】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・MM='11' のとき
- ・EaR が@-SPのとき
- ・EaM が#imm_data, @SP+, @-SPのとき

〔ニモニック〕

30 SHXL `dest`

〔命令の機能〕

shift left with extend

拡張左シフト

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図125に示す。

〔フラッグ変化〕図126に示す。

〔解説〕 `dest` を1ビット左にシフトし、元の `X_flag` の内容をLSBに詰める。MSBからあふれたビットは `X_flag` に入る。この命令は、複数ワードの1ビットシフトを行なうためのプリミティブとして専用化したものであり、シフト対象のサイズを32ビットに固定している点、1ビットのシフトしかできない点、などにおいてSHA, SHL, ROTとはかなり仕様が異なる。DIVXは被除数が多倍長の場合に使用できる命令であるが、除数が多倍長になった場合には、DIVXが使用できず、プログラムによってシフトや減算を繰り返しながら割り算を進めなければならない。その際、多倍長のシフト演算が必要になる。この命令は、このような場合に使用することを目的とした命令である。図1

107

27にこれを示す。

【0149】〔プログラム例外〕

・予約命令例外

・+= '0' のとき

・-= '1' のとき

・X= '1' のとき

・BaMXが#imm_data, @SP+, @-SPのとき

〔ニモニック〕

SHXR dest

〔命令の機能〕

shift right with extend

拡張右シフト

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図128に示す。

〔フラッグ変化〕図129に示す。

〔解説〕destを1ビット右にシフトし、元のX flagの内容をMSBに詰める。LSBからあふれたビットはX flagに入る。この命令は、複数ワードの1ビットシフトを行なうためのプリミティブとして専用化したものであり、シフト対象のサイズを32ビットに固定している点、1ビットのシフトしかできない点、などにおいてSHA, SHL, ROTとはかなり仕様が異なる。DIVXは被除数が多倍長の場合に使用できる命令であるが、除数が多倍長になった場合には、DIVXが使用できず、プログラムによってシフトや減算を繰り返しながら割り算を進めなければならない。その際、多倍長のシフト演算が必要になる。この命令は、このよ*

例：

src = H'1234

RVBY src.H, dest.H ==> dest = H'3412

RVBY src.H, dest.W ==> dest = H'34120000

RVBY src.H, dest.B ==> dest = H'34 (H'12ではない)

この命令は、endianの変換に対するオーバーヘッドを削減することを目的とした命令である。

【0151】〔プログラム例外〕

・予約命令例外

・RR= '11' のとき

・WW= '11' のとき

・EaR が@-SPのとき

・EaW が#imm_data, @SP+ のとき

〔ニモニック〕

RVBI src, dest

但し<(L2)>

〔命令の機能〕

reverse bit order

ビット順の逆転

108

*うな場合に使用することを目的とした命令である。図130にこれを示す。

【0150】〔プログラム例外〕

・予約命令例外

・+= '0' のとき

・-= '1' のとき

・X= '1' のとき

・BaMXが#imm_data, @SP+, @-SPのとき

〔ニモニック〕

RVBY src, dest

〔命令の機能〕

reverse byte order

バイト順の逆転

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図131に示す。

〔フラッグ変化〕図132に示す。

〔解説〕srcのバイト順を逆転したものをdestに転送する。srcよりもdestのサイズの方が大きい場合には、srcをdestのサイズにまでゼロ拡張した後、destのサイズでバイト順を逆転する。srcよりもdestのサイズの方が小さい場合には、srcの上位バイトをカットしてdestのサイズとした後、destのサイズでバイト順を逆転する。(srcのアドレスをずらした上でsrcとdestを同じサイズとしても、結果は同じになる)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図133に示す。

〔フラッグ変化〕図134に示す。

〔解説〕srcのビット順を逆転したものをdestに転送する。srcよりもdestのサイズの方が大きい場合には、srcをdestのサイズにまでゼロ拡張した後、destのサイズでビット順を逆転する。srcよりもdestのサイズの方が小さい場合には、srcの上位バイトをカットしてdestのサイズとした後、destのサイズでビット順を逆転する。(srcのアドレスをずらした上でsrcとdestを同じサイズとしても、結果は同じになる)

この命令は、endianの変換に対するオーバーヘッドを削減することを目的とした命令である。ビットマッ

109

ブの処理を考えると、ビットを逆順にするビットリバー
ス命令RVBIもあった方がよいが、バイトリバース命
令RVBYよりも頻度が少ないこと、追加のハードウエ*

・予約命令例外

・RR='11' のとき

・WW='11' のとき

・EaR が@-SPのとき

・EaW が#imm_data.0SP+ のとき

【0153】12-6. ビット操作命令

「本発明装置」のビット操作命令では、次の図のように
base (base address)

offset (bit address)

の2つのパラメータにより操作対象となるビットを指定
する。レジスタ上のビットを操作する場合には、このほ
かにbaseのサイズも対象ビットの指定に影響する。

〔メモリ上のビット操作をする場合〕図135に示す。

「本発明装置」の一般形のビット操作命令では、off
setの値に制限がなく、offsetがバイト境界を
越えてもよいようになっている。offsetは、符号
付き整数として扱われる。ビット操作命令では、BBフ
ィールドによりメモリアクセスの範囲を指定できるよう
になっている。これは、BTSTでreadを行なうメ
モリアドレスの範囲、およびBSET, BCLR, BN
OTで read-modify-writeを行なう
メモリアドレスの範囲を意味するものである。アクセス
されるメモリアドレスの範囲は、入出力やマルチプロセ
ッサを使用した場合に問題となることがある。しかし、
実際には、バイト単位のアクセス('B')のみが使用
できればほとんどの場合に十分であるため、ハーフワ
ード、ワード単位のアクセスは<(L2)>とする(た
だしレジスタに対するビット操作命令を除く)。また、

110

*Aが必要となる可能性が強いこと、により、RVBI命
令は<(L2)>とする。

【0152】〔プログラム例外〕

10 ハーフワード、ワード単位のアクセスが意味を持つ
のは、アラインメントのとれたハーフワードやワードをア
クセスする場合に限られているので、ハーフワード、ワ
ード単位のアクセスの機能を利用するためには、base
として必ずアラインメントのとれたアドレスを指定し
なければならないという制限を設ける。これは、アクセ
ス範囲の指定に関するインプリメントを容易にするため
である。したがって、アラインメントのとれたハーフワ
ードの単位で、対象ビットを含むメモリアクセスを行な
いたい場合には、baseとして2の倍数を指定する必
20 要がある。また、アラインメントのとれたワードの単位
で、対象ビットを含むメモリアクセスを行ないたい場合
には、baseとして4の倍数を指定する必要がある。
offsetの値については制限はない。baseとし
てアラインメントのとれていないアドレスを指定した場
合のアクセス範囲は、インプリメントに依存するものと
する。「本発明装置」では、<(L2)>となっている
メモリに対するハーフワード、ワード単位のアクセスの
インプリメントを行なう。またbaseとしてアライン
メントのとれていないアドレスを指定した場合にも、ア
30 クセス範囲はアラインメントのとれたハーフワード、ワ
ード単位でアクセスを行なう。

111

【例】

BSET.B #H'84, @H'100

offset % 8 = 4; base + offset/8 = H'110 なので、H'110 のbit4をセットする。

H'110 の1 バイトについてread-modify-write が行なわれる。

BSET.B #H'7c, @H'101

offset % 8 = 4; base + offset/8 = H'110 でアクセスサイズがバイトなので、BSET.B #H'84, @H'100 と全く同じ動作をする。

BSET.W #H'84, @H'100

offset % 8 = 4; base + offset/8 = H'110 なので、H'110 のbit4をセットする。

baseが4の倍数になっているので、H'100 ~H'103 のアラインメントのとれた32ビットについてread-modify-write を行ない、対象ビットをセットする。

BSET.W #H'7c, @H'101

offset % 8 = 4; base + offset/8 = H'110 なので、おなじようにH'110 のbit4をセットする。

しかし、baseが4の倍数ではないので、read-modify-write を行なうアクセス範囲についてはインプリメント依存である。

なお、BBで示されるサイズは、「どの範囲に対してread-modify-writeを行なうか?」ということであり、オフセットの範囲(例えば、' . B'であればオフセットが8より小さくなる、など)を規定するものではない。レジスタに対するビット操作命令では、アクセスのサイズ(baseのサイズ)によってoffset=0(MSB)のビット位置が変わるため、baseのサイズは重要な意味を持つ。baseがレジスタ直接Rnの場合は、baseのサイズ' . H' , ' . W' もく(L1))である。レジスタRnをbaseとしたビット操作命令の場合、offsetは' . B'の時下位3ビット、' . H'の時下位4ビット、' . W'の時下位5ビット、' . L'の時下位6ビットのみが有効であり、上位ビットは無視される。上位ビットが0でなくても、エラー、EITとはしない。*

【例】 BSET:Q #1, r0では、

BSETの場合にデフォルトが.Bなので、

r0.Bのビット1がセットされる。

このビットは、r0.Wのビット1とは異なったものであり、

r0.Wのビット25に対応する。

例えば、2¹⁷のビットをアクセスするつもりで

BTST #14, R0

と書くと、実際は

BTST.B #14, R0

と解釈され、offsetは上位ビットを無視するので、結局2¹のビットが対象となる。正しくは、

*キテクチャ面から見ると、上位ビットを無視するよりも、BF命令のwidthなどと同じように、きちんとoffsetの範囲をチェックする方が望ましいが、チェックを行なうことにより命令の実行時間が増大するため、offsetはアクセスサイズのビット数でmoduloをとって使用することにしている。レジスタ上に8ビットデータ、16ビットデータ、32ビットデータを置いた場合では、それぞれのデータで同じビット位置を持つビットであっても、実際には異なったビットに対応することになるので、注意が必要である。仕様が複雑化するのを避けるため、アセンブラのデフォルトはメモリ対象、レジスタ対象とも、Bとする。また、短縮形も、Bの仕様である。したがって、短縮形でアクセスできるレジスタ上の範囲は、2⁰~2⁷のビットである(図136参照)。

BTST.W #14, R0

と書かなければならない。このような場合には、アセンブラで警告を出すのが望ましいであろう。

【ニモニック】

BTST offset, base

50

【命令の機能】

113

【0154】

【数8】

-bit -> Z_flag

【0155】ビットのテスト

【命令オプション】なし

【命令ビットパターンとアセンブラ表記】図137に示す。

【フラッグ変化】図138に示す。

【解説】指定されたビットの値を反転したものをZ_flagにコピーする。EaRf, ShMfqで指定されるアドレッシングモードでは、イミディエートモード#imm_data、@-SP、@SP+は使用できない。また、Rnのモードを使用した場合、offsetの上位ビットの値は無視される。アセンブラ表記では、メモリアクセスのサイズをbaseのサイズとして指定する。BTST:Qでは、メモリアクセスのサイズは8ビットに固定されており、サイズは' . B 'のみを書くことができる。

【0156】【プログラム例外】

・予約命令例外

・RR='11' のとき

・BB='11' のとき

・EaR が@-SPのとき

・EaRf, ShMfqが#imm_data, @SP+, @-SPのとき

【ニモニック】

BSET offset, base

【命令の機能】

【0157】

【数9】

-bit -> Z_flag.1 -> bit

【0158】ビットのセット

【命令オプション】なし

【命令ビットパターンとアセンブラ表記】図139に示す。

【フラッグ変化】図140に示す。

【解説】指定されたビットの値を反転したものをZ_flagにコピーし、その後そのビットを1にセットする。EaMf, ShMfqで指定されるアドレッシングモードでは、イミディエートモード#imm_data、@-SP、@SP+は使用できない。また、Rnのモードを使用した場合、offsetの上位ビットの値は無視される。アセンブラ表記では、メモリアクセスのサイズをbaseのサイズとして指定する。BSET:Qでは、メモリアクセスのサイズは8ビットに固定されており、サイズは' . B 'のみを書くことができる。

【0159】【プログラム例外】

114

・予約命令例外

・RR='11' のとき

・BB='11' のとき

・EaR が@-SPのとき

・EaMf, ShMfqが#imm_data, @SP+, @-SPのとき

【ニモニック】

BCLR offset, base

【命令の機能】

【0160】

【数10】

-bit -> Z_flag.0 -> bit

【0161】ビットのクリア

【命令オプション】なし

【命令ビットパターンとアセンブラ表記】図141に示す。

【フラッグ変化】図142に示す。

【解説】指定されたビットの値を反転したものをZ_flagにコピーし、その後そのビットを0にクリアする。EaMf, ShMfqで指定されるアドレッシングモードでは、イミディエートモード#imm_data、@-SP、@SP+は使用できない。また、Rnのモードを使用した場合、offsetの上位ビットの値は無視される。アセンブラ表記では、メモリアクセスのサイズをbaseのサイズとして指定する。BCLR:Qでは、メモリアクセスのサイズは8ビットに固定されており、サイズは' . B 'のみを書くことができる。

【0162】【プログラム例外】

・予約命令例外

・RR='11' のとき

・BB='11' のとき

・EaR が@-SPのとき

・EaMf, ShMfqが#imm_data, @SP+, @-SPのとき

【ニモニック】

BNOT offset, base

【命令の機能】

【0163】

【数11】

-bit -> Z_flag. -bit -> bit

【0164】ビットの反転

【命令オプション】なし

【命令ビットパターンとアセンブラ表記】図143に示す。

【フラッグ変化】図144に示す。

【解説】指定されたビットの値を反転したものをZ_flagにコピーし、その後そのビットも反転する。EaMfで指定されるアドレッシングモードでは、イミディエートモード#imm_data、@-SP、@SP+は使用できない。また、Rnのモードを使用した場合、

115

offset の上位ビットの値は無視される。アセンブラ表記では、メモリアクセスのサイズを base のサイズとして指定する。

【0165】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・BB='11' のとき
- ・EaR が@-SPのとき
- ・EaMfが\$imm_data, @SP+, @-SPのとき

* 10

find first '0' or '1' in the bitfield(within a word)

0 または 1 のサーチ (1 ワード内)

〔命令オプション〕

/0 '0' をサーチ (デフォルト)

/1 '1' をサーチ

/P ビット番号の増加方向にサーチ (デフォルト)

/B ビット番号の減少方向にサーチ

<L2> (M32 でサポートする)

〔命令ビットパターンとアセンブラ表記〕図 145 に示す。

〔フラッグ変化〕図 146 に示す。

〔解説〕ワード中にある '0' または '1' のビットをサーチする。サーチを開始するビット番号 (bit offset) を offset オペランドにセットしてこの命令を実行すると、命令実行後、サーチ結果のビット番号が offset オペランドにセットされている。offset は read-modify-write となっている。offset を read-modify-write の形にしたのは、ビットの検索を繰り返して行なうことを想定したためである。サーチの行なわれるビット位置の範囲は、data オペランドの 0 ~ (data のサイズ) に限られており、ワード境界を越えることはできない。offset としてはすべてのサイズが使用可能であるが、offset の初期値の上位ビットはサーチの際には無視される。これは、上位ビットに、ワード境界を越えた分のビットオフセットや実効アドレスなどといった、別の情報が入っていることが多いと考えられるためである。また、BSCH の仕様を軽くして高速化するためである。なお、/F の時 data のサイズを表わす数、/B の時 (-1) となる。「上位ビット」とは、 \log_2 (data のビット数) よりも上位のビットを示す。data が 32 ビットであれば、 $2^5 \sim 2^{31}$ のビットが上位ビットである。サーチはビット番号の大きな方向へ向かって、つまり、big-endian の「本発明装置」では LSB の方向へ向かって行なわれるのが標準の仕様 <L0> であり、/F オプションにより実現される。逆方向のサーチ/B オ

116

*〔ニモニック〕

BSCH data, offset

〔命令の機能〕

ブションは <L2> 仕様となっている。これは、正方向のサーチと逆方向のサーチで全く別のハードウェアが必要なためである。また、サーチされる data のサイズのうち、8 ビットと 16 ビット (RR=00, 01) の指定は <L2> となっている。「本発明装置」では、<L2> となっている/B オプション、8 ビットと 16 ビットのデータサイズ (RR=00, 01) もサポートする。BSCH はビット操作命令と同じ分類にしているが、かなり異なった性質をもっている。BSCH 命令でも、他のビット操作命令と同じようにオフセットを自由に設定できる方が使いやすいという考え方があるが、その目的で BVSCCH 命令が別に設けられているため、BSCH としてはできるだけ軽い仕様に絞り、オフセットの範囲を制限している。オフセットの有効範囲は、他のビット操作命令でレジスタ直接モード Rn を指定した場合と同じ範囲である。また、一般のビット操作命令では offset が read-only、base が read-modify-write となっているのに対して、BSCH ではそれとは逆に offset が read-modify-write、data (base address) が read-only となっており、注意が必要である。BSCH/F で、指定したビットが見付からなかった場合には、サーチを行なった最後のビット (ワード境界) の次のビットのオフセットがセットされ、V_flag=1 となる。サーチ失敗時にも、EIT は起動しない。結果的に、サーチを行なったビット数の分だけオフセットが加算される。

117

【例】

@mem1 = H' 00000000, R0 = 0, big-endianで
BSCH/O/F @mem1.W, R0を実行
==>R0=0のまま、V_flagは0

@mem1 = H' ffff7fff, R0 = 0, big-endianで
BSCH/O/F @mem1.W, R0を実行
==>R0=16 となり、V_flagは0

@mem1 = H' ffffffff, R0 = 0, big-endianで
BSCH/O/F @mem1.W, R0を実行
==>R0=32 となり、V_flagは1

BSCH/Bで、指定したビットが見付からなかった場合には、offsetに(-1)がセットされる。この*

【例】

@mem1 = H' 00000000, R0 = H' 00000020 で
BSCH/O/F @mem1.W, R0.Wを実行
==>R0 = H' 00000000 となる。
(R0 = H' 00000020 のままではない)

@mem1 = H' ffff7fff, R0 = H' 00000020 で
BSCH/O/F @mem1.W, R0.Wを実行
==>R0 = H' 00000010 となる。
(R0 = H' 00000030 ではない)

@mem1 = H' ffffffff, R0 = H' 12345678 で
BSCH/O/F @mem1.W, R0.Wを実行
==>サーチ失敗のため、R0 = H' 00000020, V_flag=1となる。

@mem1 = H' ffffffff, R0 = H' 00000020 で
BSCH/O/F. @mem1.W, R0.Wを実行
==>サーチ失敗のため、V_flag=1となる。

R0はH' 00000020のまま。

R0 = H' 00000040 (上位に桁上げ伝播)ではない。

※40

- 予約命令例外

- RR='11' のとき
- MM='11' のとき
- EaR が@-SPのとき
- EaM が#imm_data, @SP+, @-SPのとき

【0167】12-7. 固定長ビットフィールド操作命令

「本発明装置」の一つのデータタイプであるビットフィールドは、ビットフィールド中のMSBの位置、および

118

*場合もV_flagがセットされるが、EITは起動されない。BSCH命令では、offsetの初期値の上位ビットは無視されるが、命令終了後にセットされるoffset値(サーチ結果)については、上位ビットまで意味を持つ値になっている。つまり、offsetの上位ビットに、ワード境界を越えた分のビットオフセットや実効アドレスなどといった別の情報が入っていても、BSCH命令実行後は、offsetの上位ビットも書き換えられてしまうわけである。サーチ成功の時はoffsetが0~31の値をとる(dataが32ビットの場合)ので、/F, /Bとも上位ビットは常に0となる。また、/Fでサーチ失敗の場合はoffset=32となるため、上位ビットが00...001に、下位ビットが00000になる。/Bでサーチ失敗の場合はoffset=(-1)となるため、上位ビットが11...111に、下位ビットが11111になる。

10

※【0166】[プログラム例外]

ビットフィールドの長さ(width)により指定される。ビットフィールドのMSBの位置は、baseとoffsetとの組で示される。baseで示されるメモリのMSB(第0bit)がoffset=0になる。

119

offsetの意味は、ビット操作命令の時と同じである。ビットフィールドとbase、offset、widthとの関係を図147に示す。

〔メモリ上のビットフィールド操作をする場合〕図147に示すように太線内が対象ビットフィールドである。固定長ビットフィールドの操作命令(BFEXT, BFEXTU, BFCMP, BFCMPU, BFINS, BFINSU)は、AI向き応用のタグ処理(タグの比較やタグの切り出し)などに特に有効である。固定長ビットフィールド命令には、次の2つのフォーマットがある。

・offsetを8ビットの一般形アドレッシングモードで指定し、widthをレジスタで指定する形式。これを':G'フォーマットと呼ぶ。':G'フォーマットでは、baseにoffset/8の値を加えることにより、実際にアクセスするメモリのアドレスが決まる。26bit以上のビットフィールドで5バイトにまたがるビットフィールドを扱うこともできる。

・offsetを8ビットのイミディエート値で指定し、widthをリテラルで指定する形式。これを':E'フォーマットと呼ぶ。':E'フォーマットでは、ワード境界を越えないビットフィールドのみを対象として速度を上げるために、baseの1ワードからはみ出した部分のビットフィールドについて、動作を保証しないものとしている。width+offset \geq sizeであってもEITは起動しないが、読みだし時、書き込み時とも値が不定となる。baseの1ワードのみのアクセスでも命令仕様を実現することができるので、offsetとは無関係に、baseのみを見て操作対象となるビットフィールドのメモリアドレスを決めることが可能である。したがって、インプリメント次第で命令実行を高速化することもできる。BF:EとBF:Gのbaseで許されるアドレッシングモードは、全く同じである。BFINS, BFINSU, BFCMP, BFCMPUでは、:G, :Eフォーマットのそれぞれに対して、さらに次の二つの形式がある。

・srcオペランドをレジスタで指定する:Rフォーマット

・srcオペランドをイミディエートで指定する:Iフォーマット

widthの値は、1~32(<<LX>)では1~64)に制限されており、命令実行前に0<width \leq

120

32(64)のチェックが行なわれる。width=0の場合もエラーである。違反した場合には、不正オペランド例外(IOE)となる。すべての命令について、offset, widthとも符号付きの数として扱われる。ただし、widthはもともと1~32(64)の値しか許されていないため、符号付きと考えるか符号なしと考えるかは実際の動作には影響せず、仕様書記述上の問題となっている。また、:Eフォーマットの命令のoffsetも符号付きとして扱われ、この場合はoffsetとして-128~+127の数を表わすことになる。(ただし、後で述べるように、:Eフォーマットではbaseのアドレスの1ワードbase~base+3からハミ出すビットフィールドについて、ハミ出した部分に関する動作を保証していない。)

BF命令のビットフィールドでない方のオペランドは、普通の整数として扱われる。したがって、例えばBFEXTの場合は、抽出されたビットフィールドがレジスタのLSB側に詰めてセットされ、MSB方向に符号拡張される。ビット位置=0(MSB)に合わせてビットフィールドをセットするのではない。baseとしてレジスタを対象とした場合には、ビットフィールドは一つのレジスタの範囲内に限られる。レジスタ対象の固定長ビットフィールド命令も「本発明装置」でサポートする。ただしこれは(<<L2>)となっている。これは、レジスタを対象としたビットフィールド操作の場合、現段階では、BF:E命令よりもシフト命令とAND命令を組み合わせて実行する方が速くなる可能性があるためである。レジスタ対象のビットフィールド命令(<<L2>))については、:Gでも次に述べる:Eと同じように、1ワード(レジスタ)からハミ出すビットフィールドについて、ハミ出した部分に関する動作を保証しないものとする。BFEXT, BFEXTUでは不定値が得られ、BFINS, BFINSUでは無視される。offset+width \geq sizeの場合もEITは起動しない。:Eフォーマットでは、対象となるビットフィールドのうちで、sizeを越えたビットオフセットを持つ部分についてのみ動作を保証していない。同様に、負のビットオフセットを持つ部分についても動作を保証していない。いずれの場合も、指定されたビットフィールドのうちで、baseのアドレスで示される1ワードに含まれる部分については、正しく実行される。

121

【例】

```
address      N-1      N
data         B' abcdefgh B' ijklmnop
```

```
      N+1
      B' qrstuvwxy
```

(a~x: 0または1)

の場合、

BFEXT: E.W #3, #9, @N, R0

==> R0 = B' lmnopqrst となる。

BFEXT: E.W #5, #9, @N, R0

==> R0 = B' ?????ijkl となる。(? は不定値)

width, src, dest のレジスタのサイズ指定は、Xフィールドによって共通に行なわれる。サイズ指定フィールドXは、32ビット演算と64ビット演算 (< < L X > >) の切り換えを行なうものであるが、具体的には次の3つの意味を持つ。

① src (dest) レジスタのサイズ指定 (: Rフォーマット)

② width レジスタのサイズ指定 (: Gフォーマット)

③ width の範囲の指定

X=0 の時 0 < width ≤ 32

X=1 の時 0 < width ≤ 64

: E : I フォーマットの場合①②は意味を持たないが、③の区別を行なうためにやはりXフィールドを使用する。すなわち、Xフィールドは、32ビットと64ビットの互換性を高めるためのビットであると言える。: I フォーマットの命令でSS≠00の時には、#i s 8のフィールドは使用しない。この時、もし#i s 8のフィールドが0になっていなくても、単に無視される。ただし、マニュアル上は、#i s 8のフィールドには0を入れるようにしておく。ビットフィールド命令のフォーマットと、それに対して使用できるサイズを列挙すると、図148、図149のようになる。ビットフィールド命

122

令についても、ビット操作命令と同じようにアクセスを行なうメモリの範囲が問題となるが、インプリメントへの依存性が強いので、強い規定は設けない。[これについては詳細仕様調整中]

[ニモニック]

BFEXT offset, width, base, dest

[命令の機能]

extract bit field (signed)

ビットフィールドの抽出 (符号付き)

[命令オプション] なし

[命令ビットパターンとアセンブラ表記] 図150に示す。

[フラグ変化] 図151に示す。

[解説] ビットフィールドの抽出を行ない結果をデスティネーションに転送する。ビットフィールドのwidthよりもデスティネーションのサイズの方が大きい時は、データが符号拡張される。また、BFEXT: Gのoffsetも符号拡張される。EaRbfのアドレッシングモードでは、@-SP, @SP+, #imm_dataのモードは使用できない。baseのレジスタ直接モードRnは<<L2>>であるが「本発明装置」ではサポートする。

[オペレーション]

123
destの初期値を {D0, D1, ..., Dd-2, Dd-1}
d=32, 64
destに設定される値を {R0, R1, ..., Rd-2, Rd-1}
d=32, 64
offset = 0, width = w
とする。offset, widthは符号付きとして扱われる。(width ≤ 0, width > dの時はエラー)
この時、抽出される部分のビットフィールドとフラグの変化は、次のようになる。

(d ≥ w の時)

baseのbit0

↓

[... B0, B1, ..., B0-2, B0-1, B0, B0+1, ..., B0+w-2, B0+w-1, B0+w, B0+w+1, ...]

この部分を符号拡張してdestに設定する

[B0, B0+1, ..., B0+w-2, B0+w-1] ==>

[B0, B0, ..., B0, B0, B0+1, ..., B0+w-2, B0+w-1] ==>

d-w ビットだけ符号拡張される

[R0, R1, ..., Rd-w-1, Rd-w, Rd-w+1, ..., Rd-2, Rd-1] (destに設定される)

(d < wの時)

「本発明装置」 32 では起り得ないケースである。

baseのbit0

↓

[... B0, B1, ..., B0-1, B0, B0+1, ..., B0+w-d-1, B0+w-d, ..., B0+w-2, B0+w-1, B0+w, ...]

この部分はカットされる この部分をdestに設定する

[B0, B0+1, ..., B0+w-d-1, B0+w-d, ..., B0+w-2, B0+w-1] ==>

[B0+w-d, ..., B0+w-2, B0+w-1] ==>

この部分がカットされる

[R0, ..., Rd-2, Rd-1] (destに設定される)

M_flag R0

または (d ≥ w の時) B0

(d < w の時) B0+w-d

Z_flag [R0 ~d-1] = 0

または (d ≥ w の時) [B0 ~0+w-1] = 0

(d < w の時) [B0+w-d ~0+w-1] = 0

V_flag☆ S[B0 ~0+w-1] < -2^(d-1).or.

125

$$S[Bo \sim o+w-1] \geq +2^{(d-1)}$$

または ($d \geq w$ の時) 0

($d < w$ の時) $Bo=Bo+1=\dots=Bo+w-d-1=Bo+w-d$ の時ク

リア、それ以外の場合にセットとなる。

126

「本発明装置」32であれば常にクリアされる。

【0168】 (プログラム例外)

・予約命令例外

・RR='11' のとき

・+='0' のとき

・X='1' のとき

・EaR が@-SPのとき

・EaRbf が#imm_data, @SP+, @-SPのとき

・不正オペランド例外

・width ≤ 0 , width > 32 のとき

〔ニモニック〕

BFEXTU offset, width, base, dest

〔命令の機能〕

extract bit field (unsigned)

ビットフィールドの抽出 (符号なし)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図152に示す。

〔フラッグ変化〕図153に示す。

〔解説〕ビットフィールドの抽出を行ない結果をデスティネーションに転送する。ビットフィールドのwidthよりデスティネーションのサイズの方が大きい時は、データがゼロ拡張される。ただし、BFEXTU: Gのoffsetは符号拡張される。EaRbfのアドレッシングモードでは、@-SP, @SP+, #imm*baseのbit0

↓

[... B0. B1.... B0-2. B0-1. B0. B0+1.... B0+w-2. B0+w-1. B0+w. B0+w+1....]

この部分をゼロ拡張してdestに設定する

[B0. B0+1.... B0+w-2. B0+w-1] ==>

[0. 0..... 0. B0. B0+1.... B0+w-2. B0+w-1] ==>

d-wビットだけゼロ拡張される

[R0. R1... Rd-w-1. Rd-w. Rd-w+1.... Rd-2. Rd-1] (destに設定される)

($d < w$ の時)

「本発明装置」32では起り得ないケースである。

*_dataのモードは使用できない。baseのレジスタ直接モードRnは<L2>であるが「本発明装置」ではサポートする。

〔オペレーション〕

destの初期値を [D0. D1.... Dd-2. Dd-1]

d=32, 64

destに設定される値を[R0. R1.... Rd-2. Rd-1]

d=32, 64

offset = 0, width = w

とする。offset, widthは符号付きとして扱われる。(width ≤ 0 , width $> d$ の時はエラー)

この時、抽出される部分のビットフィールドとフラッグの変化は、次のようになる。

($d \geq w$ の時)

127
baseのbit0

128

↓
[...Bo, B1, ..., Bo-1, Bo, Bo+1, ..., Bo+w-d-1,

この部分はカットされる

Bo+w-d, ..., Bo+w-2, Bo+w-1, Bo+w...]

この部分をdestに設定する

[Bo, Bo+1, ..., Bo+w-d-1, Bo+w-d, ..., Bo+w-2, Bo+w-1] ==>

[Bo+w-d, ..., Bo+w-2, Bo+w-1] ==>

この部分がカットされる

[R0, ..., Rd-2, Rd-1]

(destに設定される)

M_flag R0

または (d > w の時) 0

(d = w の時) Bo

(d < w の時) Bo+w-d

Z_flag [R0 ~ d-1] = 0

または (d ≥ w の時) [Bo ~ o+w-1] = 0

(d < w の時) [Bo+w-d ~ o+w-1] = 0

V_flag☆ U[Bo ~ o+w-1] ≥ +2^d

または (d ≥ w の時) 0

(d < w の時) Bo=Bo+1= ... =Bo+w-d-1=0 の時クリ

ア、それ以外の場合にセットとなる。

「本発明装置」32であれば常にクリアされる。

【0169】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・+= '0' のとき

・X = '1' のとき

・EaR が@-SPのとき

・EaRbf が#imm_data, @SP+, @-SPのとき

・不正オペランド例外

・width ≤ 0, width > 32のとき

〔モニタック〕

BFINS src, offset, width, b

ase

〔命令の機能〕

insert bit field

ビットフィールドの挿入 (符号付き)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図154に示 50 扱われる。(width ≤ 0, width > d の時はエ

す。

〔フラグ変化〕図155に示す。

〔解説〕ソースの値をビットフィールドに挿入する。ソースのサイズよりもビットフィールドのwidthの方が大きい時は、データが符号拡張される。また、BFINS:Gのoffsetも符号拡張される。EaRbfのアドレッシングモードでは、@-SP, @SP+, #imm_dataのモードは使用できない。baseのレジスタ直接モードRnは<(L2)>であるが「本発明装置」ではサポートする。

〔オペレーション〕

srcの初期値を [S0, S1, ..., Ss-2, Ss-1]

s=8, 16, 32, 64(:I)

s=32, 64(:R)

offset = o, width = w

とする。offset, widthとも符号付きとして扱われる。(width ≤ 0, width > d の時はエ

129

130

ラー)

* ($w \geq s$ の時)

この時、挿入される部分のビットフィールドとフラッグ
の変化は、次のようになる。

ビットフィールドの変化

*

baseのbit0

↓

$$[\dots B0. B1. \dots B_{o-1}. B_o. B_{o+1}. \dots B_{o+w-s-1}. B_{o+w-s}. B_{o+w-s+1}. \dots B_{o+w-1}. B_{o+w}$$

$$\dots]] \Rightarrow$$

$$[\dots B0. B1. \dots B_{o-1}. \underline{S0. S0. \dots S0. S0. S1. \dots S_{s-1}. B_{o+w}}$$

$$\text{src が } w-s \text{ ビットだけ符号拡張される} \dots]]$$
($w < s$ の時)

ビットフィールドの変化

baseのbit0

↓

$$[\dots B0. B1. \dots B_{o-2}. B_{o-1}. B_o. B_{o+1}. \dots B_{o+w-1}. B_{o+w} \dots]] \Rightarrow$$

$$[\dots B0. B1. \dots B_{o-2}. B_{o-1}. S_{s-w}. S_{s-w+1}. \dots S_{s-1}. B_{o+w} \dots]]$$
src の $[S0. S1. \dots S_{s-w-1}]$ がカットされる。M_flag 対象ビットフィールドのMSB(B_o)の変化を基準とする。または ($w \geq s$ の時) $S0$ ($w < s$ の時) S_{s-w} Z_flag 対象ビットフィールド $[B_o \sim B_{o+w-1}]$ の変化を基準とするまたは ($w \geq s$ の時) $[S0 \sim S_{s-1}] = \text{src} \cdot 0$ ($w < s$ の時) $[S_{s-w} \sim S_{s-1}] = 0$ V_flag☆ $S[S0 \sim s-1] = \text{src} < -2^{(w-1)}. \text{or.}$ $S[S0 \sim s-1] = \text{src} \geq +2^{(w-1)}$ または ($w \geq s$ の時) 0($w < s$ の時) $S0=S1=\dots=S_{s-w-1}=S_{s-w}$ の時クリ

ア、それ以外の場合にセットとなる。

【0170】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・+= '0' のとき

・X = '1' のとき

・SS='11' のとき

・EaR が@-SPのとき

・EaMbf が#imm_data.@SP+,@-SPのとき

・不正オペランド例外

・width ≤ 0 , width > 32 のとき

〔ニモニック〕

BFINSU src, offset, width,
base

〔命令の機能〕

40 insert bit field

ビットフィールドの挿入 (符号なし)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図156に示す。

〔フラッグ変化〕図157に示す。

〔解説〕ソースの値をビットフィールドに挿入する。ソースのサイズよりもビットフィールドのwidthの方が大きい時は、データがゼロ拡張される。一方、BFINSU:Gのoffsetは符号拡張される。EaRb

50 fのアドレッシングモードでは、@-SP, @SP+,

131

#imm_dataのモードは使用できない。base
のレジスタ直接モードRnは〈〈L2〉〉であるが「本
発明装置」ではサポートする。

[オペレーション]

srcの初期値を [S0, S1, ..., Ss-2, Ss-1]

s=8, 16, 32, 64(:1)

s=32, 64(:R)

offset = 0, width = w

* 10

baseのbit0

↓

[... B0, B1, ..., Bo-1, Bo, Bo+1, ..., Bo+w-s-1, Bo+w-s, Bo+w-s+1, ..., Bo+w-1, Bo+w

...]] ==>

[... B0, B1, ..., Bo-1, 0, 0, ..., 0, S0, S1, ..., Ss-1, Bo+w

srcがw-s ビットだけ符号拡張される

...]]

(w < s の時)

ビットフィールドの変化

baseのbit0

↓

[... B0, B1, ..., Bo-2, Bo-1, Bo, Bo+1, ..., Bo+w-1, Bo+w...]] ==>

[... B0, B1, ..., Bo-2, Bo-1, Ss-w, Ss-w+1, ..., Ss-1, Bo+w...]]

src の[S0, S1, ..., Ss-w-1] がカットされる。

M_flag 対象ビットフィールドのMSB(Bo) の変化を基準とする。

または (w > s の時) 0

(w = s の時) S0

(w < s の時) Ss-w

Z_flag 対象ビットフィールド[Bo ~ o+w-1] の変化を基準とする

または (w ≥ s の時) [S0 ~ s-1] = src = 0

(w < s の時) [Ss-w ~ s-1] = 0

V_flag☆ U[S0 ~ s-1] = src ≥ + 2^{-w}

または (w ≥ s の時) 0

(w < s の時) S0=S1=...=Ss-w-1=0 の時クリア

、それ以外の場合にセットとなる。

[0171] [プログラム例外]

132

*とする。offset, widthとも符号付きとして
扱われる。(width ≤ 0, width > d の時はエ
ラー)

この時、挿入される部分のビットフィールドとフラッグ
の変化は、次のようになる。

(w ≥ s の時)

ビットフィールドの変化

133

・予約命令例外

- ・RR='11' のとき
- ・+= '0' のとき
- ・X = '1' のとき

- ・SS='11' のとき
- ・EaR が@-SPのとき
- ・EaMbf が#imm_data, @SP+, @-SPのとき

・不正オペランド例外

- ・width ≤ 0, width > 32のとき

〔ニモニック〕

BFCMP src, offset, width, base

〔命令の機能〕

compare bit field (signed)
ビットフィールドの比較 (符号付き)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図158に示す。

〔フラッグ変化〕図159に示す。

〔解説〕ソースsrcの値とビットフィールドの値を比較する。ソースのサイズとビットフィールドのwidthの値が異なっている場合は、サイズの小さい方のデータ * baseのbit0

↓

[... B0, B1, ..., B0-2, B0-1, B0, B0+1, ..., B0+w-2, B0+w-1, B0+w, B0+w+1, ...]

この部分を符号拡張してsrcと比較する

(s < wの時)

baseのbit0

↓

[... B0, B1, ..., B0-1, B0, B0+1, ..., B0+w-s-1, B0+w-s, ..., B0+w-2, B0+w-1, B0+w, ...]

srcを符号拡張してこの部分と比較する

L_flag S[B0 ~ 0+w-1] - S[S0 ~ s-1] < 0

比較結果によってセットされる。

40

Z_flag S[B0 ~ 0+w-1] - S[S0 ~ s-1] = 0

比較結果によってセットされる。

【0172】〔プログラム例外〕

134

*タが符号拡張されてから比較される。また、BFCMP: Gのoffsetも符号拡張される。EaRbfのアドレッシングモードでは、@-SP, @SP+, #imm_dataのモードは使用できない。baseのレジスタ直接モードRnは<<L2>>であるが「本発明装置」ではサポートする。

〔オペレーション〕

srcの初期値を [S0, S1, ..., Ss-2, Ss-1]

s=8, 16, 32, 64(:1)

s=32, 64(:R)

offset = 0, width = w

とする。offset, widthとも符号付きとして扱われる。(width ≤ 0, width > dの時はエラー)

この時、比較される部分のビットフィールドとフラッグの変化は、次のようになる。

(s ≥ wの時)

・予約命令例外

- ・RR='11' のとき
- ・+= '0' のとき
- ・- = '1' のとき
- ・SS='11' のとき
- ・EaR が@-SPのとき
- ・EaRbf が#imm_data, @SP+, @-SPのとき

・不正オペランド例外

- ・width ≤ 0, width > 32のとき

50 〔ニモニック〕

135

BFCMPU src, offset, width,
base

〔命令の機能〕

compare bit field (unsigned)

ビットフィールドの比較 (符号なし)

〔命令オプション〕 なし

〔命令ビットパターンとアセンブラ表記〕 図160に示す。

〔フラッグ変化〕 図161に示す。

srcの初期値を

[S0, S1, ..., Ss-2, Ss-1]

s=8, 16, 32, 64(:I)

s=32, 64(:R)

offset = o, width = w

とする。offset, widthとも符号付きとして扱われる。(width ≤ 0, width > dの時はエラー)

baseのbit0

↓

[... B0, B1, ..., Bo-2, Bo-1, Bo, Bo+1, ..., Botw-2, Botw-1, Botw, Botw+1, ...]

この部分をゼロ拡張してsrcと比較する

(s < wの時)

baseのbit0

↓

[... B0, B1, ..., Bo-1, Bo, Bo+1, ..., Botw-s-1, Botw-s, ..., Botw-2, Botw-1, Botw, ...]

..]

srcをゼロ拡張してこの部分と比較する

L_flag U[Bo ~ Botw-1] - U[S0 ~ Ss-1] < 0

比較結果によってセットされる。

Z_flag U[Bo ~ Botw-1] - U[S0 ~ Ss-1] = 0

比較結果によってセットされる。

【0173】〔プログラム例外〕

・予約命令例外

・RR='11'のとき

・+='0'のとき

・-='1'のとき

・SS='11'のとき

・EaRが@-SPのとき

・EaRbfが#imm_data, @SP+, @-SPのとき

・不正オペランド例外

・width ≤ 0, width > 32のとき

136

*〔解説〕 ソースsrcの値とビットフィールドの値を比較する。ソースのサイズとビットフィールドのwidthの値が異なっている場合は、サイズの小さい方のデータがゼロ拡張されてから比較される。一方、BFCMPU:Gのoffsetは符号拡張される。EaRbfのアドレッシングモードでは、@-SP, @SP+, #imm_dataのモードは使用できない。baseのレジスタ直接モードRnは〈L2〉であるが「本発明装置」ではサポートする。

*10 〔オペレーション〕

※この時、比較される部分のビットフィールドとフラッグの変化は、次のようになる。

※ (s ≥ wの時)

【0174】12-8. 任意長ビットフィールド操作命令

任意長のビットフィールド操作命令には、次のようなものがある。

一般的な演算と転送

BVMAP

転送

BVCPY

繰り返しパターンの演算と転送

BVPAT

0または1のサーチ

BVSCH

BVMAP, BVPAT, BVCPYは、ビットマップディスプレイ上のウインドウ操作(bit blt)を主な目的とした命令である。説明のため、ビットマップディスプレイの属性について用語を定義する。

(color scale, color offsetとbit-dot極性)

・color scale : 1dotを連続の何bitで表すか。

137

例:

`<color scale = 1>`

1bit が 1 dot。1 バイトで連続した 8 dot。

白黒のビットマップディスプレイ。

または、color を構成する各ビットをバンクにしたビットマップ
ディスプレイ。`<color scale = 4>`

隣合った 4 bit で 1 dot。1 バイトで連続した 2 dot。

16色カラーのビットマップディスプレイ。

・ bit-dot 極性

これは、ビットマップディスプレイとプロセッサとの組み合わせに対して生ずる概念である。小さいアドレスの方が左側に表示されるような一般的なビットマップディスプレイにおいて、小さいビット番号に対応するドットも左側に表示される場合、このようなビットマップディスプレイを正の bit-dot 極性を持つという。また、逆のものを負の bit-dot 極性を持つという。つまり、big-endian のプロセッサでは、MSB ビットが左側に表示される場合に正の bit-dot 極性を持つことになる。

・ color offset

$$\text{bit offset} = X * \text{color scale} + \text{color offset}$$

負の bit-dot 極性の時

$$\text{bit offset} = (X * \text{color scale} + \text{color offset}) . \text{xor} . 7$$

ところで、実際の「本発明装置」の BVMAP, BVCPY, BVPA T 命令では、インプリメントのことを考えて制約を設け、次のような場合にのみ使用できるようになっている。

・ bit-dot 極性が正

・ color scale が 1

このため、ビットマップディスプレイのハードウェアをある程度規定することはやむを得ない。具体的な制限は次のようになる。

・ bit-dot 極性が正なので、「本発明装置」を big-endian とした場合には、アドレスの小さい方、ビット番号の小さい方 (MSB) が画面の左側に表示されなければならない。

・ color scale = 1 のみなので、color scale ≠ 1 のビットマップディスプレイに対しては次のような制約がある。color scale ≠ 1 のビットマップディスプレイでは、color offset 毎に演算の種類を変えることができなくなる。BVMAP 命令で color scale の変更をすることはできないので、ビットマップディスプレイの color scale が 1 でない時には、内部表現も同じ color scale にしないと、BVMAP 命令は、使えない。その場合、画面イメージの内部表現がハードウェアに依存することになるので、他のハードウェアと

138

* 1 dot を構成する複数のビットのうち、第何ビットを操作するか、ということ。

$$0 \leq \text{color offset} < \text{color scale}$$

という関係が成り立つ。これは、ビットマップディスプレイハードウェアの属性ではなく、ビットマップディスプレイ操作上のパラメータである。base address に対応するドットから、横方向に X (dot offset) だけ変位したドットを操作する場合、そのメモリ上の bit offset は次のように計算される。(dot offset は画面上の点の概念、bit offset はメモリ上の bit の概念である。)

* 正の bit-dot 極性の時

の間でデータの転送を行なう時は、データ形式の変換が必要になる。任意長ビットフィールド操作命令はオペランドが多く、実行時間も長い。したがって、実行中での割り込み受け付けや、割り込み処理後の再実行のメカニズムが必要である。「本発明装置」では、オペランドの指定と演算の進行状況の表現のために固定番号のレジスタを使用している。そのため、任意長ビットフィールド命令実行中に割り込みが入っても、割り込み処理ハンドラ中でそのレジスタの退避と復帰が正しく行なわれていれば、割り込み処理後に、そのビットフィールド命令を途中から再開できる。実行中断後に状態の退避やコンテキストスイッチを行なったり、コンテキストスイッチ後に別のプロセスで同じビットマップ命令を実行し、再び前のコンテキストに戻って前のビットマップ命令を再開したとしても、問題なく動かなければならない。また、BTRON の仕様では、VRAM ではない普通のメインメモリに対しても、文字や図形の描画が行なわれることがある。したがって、任意長ビットフィールド命令でもページフォルトの起こる可能性があり、ストリング命令と同様に、ページフォルトによる実行中断にも対処できなければならない。BVMAP, BVCPY 命令では、インサートエディタなどで真横に図形を移動することを考え、ビットマップのソースとデスティネーションのオーバーラップにも対応できるようになっている。具

139

体的には、ストリング命令と同じように、演算を行なう方向を命令中のオプション／F、／Bとして指定する。ソフトウェアにより適当な方向を判定し、デスティネーションがソースを破壊しないように演算を進める。ただし、インプリメントの負担を考え、逆方向の処理を指定するオプション／Bは〈〈L2〉〉となっている。「本発明装置」ではBTRONの動作を高速化するための逆方向処理もサポートする。srcとdestがオーバーラップしていた場合、srcのbase～offsetよりもdestのbase～offsetの方が小さければ、offsetの小さい方から処理することによってdestがsrcを破壊することなく処理を進めることができる。この目的で／Fオプションを使用する。画面とビットマップとの対応は、通常オフセット（アドレス）の小さい方が左側になる。したがって、srcよりもdestのbase～offsetの方が小さくなるのは、文字の削除などによってビットマップデータを左に動かそうとした時である。また、srcのbase～offsetよりもdestのbase～offsetの方が大きければ、offsetの大きい方から処理することによってdestがsrcを破壊することなく処理を進めることができる。この目的で／Bオプションを使用する。srcよりもdestのbase～offsetの方が大きくなるのは、文字の挿入などによってビットマップデータを右に動かそうとした時である。srcとdestがオーバーラップする可能性がある場合には、ソフトウェアの判断により正しいオプションを使用し、destがsrcを破壊しないように演算を進める必要がある。ただし、／Bオプションは〈〈L2〉〉となっているので、／Bが使用できない場合には、srcを一旦他の場所にコピーしてからdestとの演算を行わなければならない。オーバーラップがない場合には、どちらのオプションを使っても結果は変わらない。ここで、問題は、destのbase～offsetの方が小さいのに／Bオプションを使用した場合や、destのbase～offsetの方が大きいのに／Fオプションを使用した場合にどのような動作を行なうかということである。基本的には、既に演算の終わった部分のdestが、srcのまだ参照されていない部分を破壊する形になるため、正しい結果が得られない。しかも、この時は、アルゴリズム上、命令が途中で中断して再実行を行なった場合に、結果が変わってくることがある。もともと正しい結果を保証していないのであるから、実行中断によって結果が変わったとしても構わないはずであるが、実行中断のなかった場合は正しい結果が得られることもあるので、再現不可能なバグが入りやすい状況になる。しかし、このエラーチェックをきちんと行なう

140

とオーバーヘッドが増え、実行時間の低下をもたらすので、エラーチェックは行なわない。ユーザの側で注意が必要である。任意長ビットフィールド命令では、レジスタ上のビットオフセットoffset、ビット幅width、パターンデータpatternのサイズとして、32ビットまたは64ビット〈〈LX〉〉のみが使用できる。8、16ビットの指定は行なわない。32ビットと64ビットのレジスタサイズの選択は、Xフィールドによって共通に行なわれる。BVMAP、BVCPY、BVPAT命令のdest側のメモリアクセス方法については、writeまたはread-modify-writeということによって特に規定しない。BV命令でwidth≤0の場合には、何もせずに命令を終了し、EITとはしない。この時、BVSCH命令では、長さによる終了を示すV_flag（サーチ失敗と同じ）がセットされる。これは、BV命令やストリング命令の様な高機能命令の場合には、その外側でさらに高機能のサブルーチンを作ることが多く、チェックが必要であればそのサブルーチンで行なえばよいと考えているためである。例えば、BVMAPであれば、それをライン数だけ繰り返してBitBlt関数を作ることが多く、その時はwidthがすべて共通になるので、毎回widthをチェックする必要はない。一方、BF命令のように、コンパイラが直接生成する可能性のあるコードでは、できるだけチェックを厳重にする必要がある。したがって、BF命令のwidthは例外で検出するようにしている。任意長ビットフィールド命令でoffset+widthがオーバーフローする場合には、割り込みによる命令実行中断時、および命令終了時のレジスタ上のoffset値がおかしな値になり、正常な命令の実行ができなくなる。この場合は、動作を保証しないものとする。アーキテクチャ上は、命令実行開始時にこれを検出して不正オペランド例外（IOE）とするのが望ましいが、チェックのために実行時間が伸びるので、チェックせずに実行するものとする。（なお、ストリング命令の場合は、offsetに相当するのが整数ではなくポインタアドレスとなっているため、オーバーフローとしては扱わず、単にアドレスがラップアラウンドするだけである。）

〔ニモニック〕

BVSCH

〔命令の機能〕

find first '0' or '1' in the
bitfield (variable length)
0または1のサーチ（任意長ビットフィールド）

〔命令オプション〕

141
/0 '0' をサーチ (デフォルト)
/1 '1' をサーチ

/F ビット番号の増加方向にサーチ (デフォルト)
/B ビット番号の減少方向にサーチ

<<L2>> (「本発明装置」ではサポートする)

〔命令ビットパターンとアセンブラ表記〕図162に示す。

〔フラッグ変化〕図163に示す。

〔解説〕任意長のビットフィールドの中にある'0'または'1'のビットをサーチする。サーチを開始するビット番号 (bit offset) をoffsetオペランド (R1) にセットしてこの命令を実行すると、命令実行後、サーチ結果のビット番号がoffsetオペランド (R1) にセットされている。つまり、offsetはread-modify-writeとなっている。これは、ビットの検索を繰り返して行なうことを想定したためである。offsetは符号付き整数として扱われ、その値は任意である。BVSCCHを実行した結果、サーチ失敗だった場合には、V_flagをセットし、offsetは次にサーチすべきビットを指示。EITは起動しない。BVSCCH命令のオフセットやV_flagの変化方法は、BSCH命令に準じたものである。/Bによる逆方向のサーチは<<L2>>仕様となっているが「本発明装置」ではサポートする。この命令は、ディスクやメモリの空きブロック検索などに使用することを目的としたものである。なお、任意長ビットフィールド命令やストリング命令などの高機能命令の詳細仕様や、命令終了後のレジスタ値については、付録11を参照のこと。

【0175】〔プログラム例外〕

・予約命令例外

- ・+= '0' のとき
- ・X= '1' のとき
- ・P= '1' のとき

〔ニモニック〕

BVMAP

〔命令の機能〕

bit operation (one line Bit Blt)

ビットマップ演算

〔命令オプション〕

/F offsetの小さい方から処理する (デフォルト)

/B offsetの大きい方から処理する

<<L2>> (「本発明装置」ではサポートする)

142

〔命令ビットパターンとアセンブラ表記〕図164に示す。

〔フラッグ変化〕図165に示す。

〔解説〕スクリーン上のビットマップ操作を行なうために、任意長のビットフィールドsrc, destに対する各種の論理演算をする命令である。演算の種類はR5の下位4ビットで指定され、次の16種類が用意されている。

【0176】

〔数12〕

T	True	1 ==> dest
F	False	0 ==> dest
ND	NotDest	~dest ==> dest
D	Dest	dest ==> dest
NS	NotSrc	~src ==> dest
S	Src	src ==> dest
A	And	dest .and. src ==> dest
O	Or	dest .or. src ==> dest
X	Xor	dest .xor. src ==> dest
NA	NotAnd	~dest .and. src ==> dest
NO	NotOr	~dest .or. src ==> dest
AN	AndNot	dest .and. ~src ==> dest
ON	OrNot	dest .or. ~src ==> dest
NAN	NotAndNot	~dest .and. ~src ==> dest
NON	NotOrNot	~dest .or. ~src ==> dest
NX	NotXor	~dest .xor. src ==> dest

【0177】このうちD (Dest) の演算モードは、

対称性のために設けられている。ニモニックと実際のビットパターンとの対応については、付録を参照のこと。演算を指定するレジスタR5の上位ビットが0でない場合にも、特にチェックは行なわないものとする。ただし、チェックが行なわれていなくても、上位ビットには必ず'0'を入れてもらうように、マニュアル等で指導する必要がある。不正オペランド例外 (IOE) としないのは、インプリメントの負担が大きく、実行速度に影響が出るためである。/F, /Bオプションは、offsetの小さい方から処理するか、offsetの大きい方から処理するかを指定する。これは、ビットマップ

143

のsrcとdestがオーバーラップしていた場合に、処理の方向を明確にしておかないと、destがsrcを破壊して正しい結果が得られないからである。srcとdestがオーバーラップしていた場合、srcのbase～offsetよりもdestのbase～offsetの方が小さければ、offsetの小さい方から処理することによってdestがsrcを破壊することなく処理を進めることができる。この目的で/Fオプションを使用する。画面とビットマップとの対応は、通常オフセット（アドレス）の小さい方が左側になる。したがって、srcよりもdestのbase～offsetの方が小さくなるのは、文字の削除などによってビットマップデータを左に動かそうとした時である。また、srcのbase～offsetよりもdestのbase～offsetの方が大きければ、offsetの大きい方から処理することによってdestがsrcを破壊することなく処理を進めることができる。この目的で/Bオプションを使用する。srcよりもdestのbase～offsetの方が大きくなるのは、文字の挿入などによってビットマップデータを右に動かそうとした時である。なお、destのbase～offsetの方が小さいのに/Bオプションを使用した場合や、destのbase～offsetの方が大きいのに/Fオプションを使用した場合には、結果(dest)を保証しないものとする。特に、このような場合には、命令実行中に割り込みやページフォールトなどが発生して命令再実行が起こると、結果が変わってくることもある。srcとdestがオーバーラップする可能性がある場合には、ソフトウェアの判断により正しいオプションを使用し、destがsrcを破壊しないように演算を進める必要がある。ただし、/Bオプションは〈〈L2〉〉となっているので、/Bが使用できない場合には、srcを一旦他の場所にコピーしてからdestとの演算を行わなければならない。「本発明装置」では/Bオプションはサポートする。オーバーラップがない場合には、どちらのオプションを使っても結果は変わらない。

←base～offsetが小 base～offsetが大→

[オーバーラップなし] 図166に示す。

[オーバーラップあり—destのbase～offsetが小]

図167に示す。

[オーバーラップあり—destのbase～offsetが大]

図168に示す。

【0178】〔プログラム例外〕

144

・予約命令例外

・Q='1' のとき

・X='1' のとき

・P='1' のとき

〔ニモニック〕

BVCPY

〔命令の機能〕

10 bit transfer

ビットマップ転送

〔命令オプション〕

/F offsetの小さい方から処理する（デフォルト）

/B offsetの大きい方から処理する

<<L2>>（「本発明装置」ではサポートする）

〔命令ビットパターンとアセンブラ表記〕 図169に示す。

〔フラッグ変化〕 図170に示す。

20 〔解説〕スクリーン上のビットマップ操作を行なうために、任意長のビットフィールドsrc、destの間の転送をする命令である。この命令は、BVMAP命令から演算の機能ははずして転送のみに限定し、高速化を目指したものである。/F、/Bオプションの意味はBVMAPと同じである。ビットマップのsrcとdestがオーバーラップしていなければ、どちらのオプションを使っても結果は変わらないが、srcとdestがオーバーラップした場合には、ソフトウェアの判断により正しいオプションを使用し、destがsrcを破壊しないように演算を進める必要がある。/Bオプションの場合、R1、R4に入れるオフセット値としては、転送の対象となるビットフィールドの最大+1のオフセット値を指定する。これは、SMOV/B、SCMP/Bの仕様との対応を考えたものである。/Bオプションは〈〈L2〉〉であるが「本発明装置」ではサポートする。

【0179】〔プログラム例外〕

・予約命令例外

・Q='1' のとき

・X='1' のとき

・P='1' のとき

〔ニモニック〕

BVPAT

〔命令の機能〕

cyclic bit operation

パターンとビットマップの演算

〔命令オプション〕なし

50 〔命令ビットパターンとアセンブラ表記〕 図171に示す。

す。

〔フラッグ変化〕図172に示す。

〔解説〕スクリーン上のビットマップをあるパターンで埋めたり、スクリーン上のビットマップとあるパターンの演算を行ないたい場合に使用する命令である。patternを繰り返し発生しながら、ビットフィールドとの論理演算を行なう。演算指定(R5)の上位ビットが0でない場合は単に無視され、特にチェックは行なわないものとする。ただし、チェックが行なわれていなくても、将来の拡張のため、上位ビットには必ず'0'を入*10

・予約命令例外

- ・+='0' のとき
- ・X='1' のとき
- ・P='1' のとき

【0181】12-9. 10進演算命令

10進演算に関しては、符号なしPACKED形式(BCD)の10進数の1ワードの加減算とPACK/UNPACK処理をメインプロセッサの<(L1)>仕様としてサポートし、符号付きPACKED形式10進数の1ワードの加減算を<(L2)>仕様としてサポートする。また、多桁の10進数の加減乗除はコプロセッサで行なう。このうち、本章では符号なしPACKED形式10進数の加減算とPACK/UNPACK処理について説明を行なう。符号付きPACKED形式10進数をサポートする<(L2)>の命令については、後の章で説明を行なう。10進演算のアドレッシングモードは一般命令と同じになっている。本発明装置では本節で述べる10進演算命令4種類はサポートしない。

〔ニモニック〕

ADDDX src, dest (本発明装置ではサポートしない)

〔命令の機能〕

dest + src + X_flag ==> dest BCD

BCDの加算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図173に示す。

〔フラッグ変化〕図174に示す。

〔解説〕パックされたBCDの加算を行なう。8ビット(2桁)、16ビット(4桁)、32ビット(8桁)、64ビット(16桁)のBCDデータを扱うことができる。ただし、64ビットは<(LX)>仕様である。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースがゼロ拡張された上で加算される。BCDの数は符号拡張が無意味なので、基本的には符号なしの数と考え、ADDDXのフラッグ変化はADDUに準じるものとする。結果がdestに入らない時にV_flagがセットされること、d

*れてもらうように、マニュアル等で指導する必要がある。不正オペランド例外(IOE)としないのは、インプリメントの負担が大きく、実行速度に影響が出るためである。この命令では、BVMAP、BVCOPYとは異なり、書き込みの際にシフトは行なわない。offsetの指定は、単にパターンをクリッピングするだけである。(これに対して、BVMAP命令では、srcとdestのoffsetがずれていた場合にはシフトが行なわれる)

【0180】〔プログラム例外〕

<sの時はdestのサイズからの桁上げがX_flagにセットされること、などもADDUと同様である。ただし、ADDUとは異なり、Z_flagはADDX, SUBXのように累積で変化する。src, destの各桁が0~9以外の数を含んでいた場合、つまりADDDX, SUBDXのオペランドがBCDでなかった場合には、EITとはならないが、destやフラッグに設定される結果は保証できない(インプリメント依存になる)ものとする。不正オペランド例外(IOE)としないのは、インプリメントの負担が大きく、実行速度に影響が出るためである。

【0182】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・MM='11' のとき

・EaR が@-SPのとき

・EaM が#imm_data, @SP+, @-SPのとき

・<(L1)>機能例外

・ADDDXの正しいビットパターンがデコードされたとき

〔ニモニック〕

40 SUBDX src, dest (本発明装置ではサポートしない)

〔命令の機能〕

dest - src - X_flag ==> dest BCD

10進BCDの減算

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図175に示す。

〔フラッグ変化〕図176に示す。

〔解説〕パックされたBCDの減算を行なう。8ビット(2桁)、16ビット(4桁)、32ビット(8桁)、

147

64ビット(16桁)のBCDデータを扱うことができる。ただし、64ビットは〈〈LX〉〉仕様である。ソースオペランドのサイズがデスティネーションオペランドのサイズよりも小さい時は、ソースがゼロ拡張された上で加算される。BCDの数は符号拡張が無意味なので、基本的には符号なしの数と考え、SUBDXのフラグ変化はSUBUに準じるものとする。結果が負になった時にV_flagがセットされること、d<sの時はdestのサイズからの桁下げがX_flagにセットされること、などもSUBUと同様である。ただし、SUBUとは異なり、Z_flagはADDX, SUBXのように累積で変化する。SUBDXで結果が負になった場合には、destは絶対値表現ではなく補数表現(10の補数)となる。したがって、destは上位桁からの繰り下がりがあった場合と同じ値になる。

例: 16ビットでSUBDXを実行する場合

```
dest  src
0123 - 0456 = (-0333)
destは(-333) = 9667 となる。
```

src, destの各桁が0~9以外の数を含んでいた場合、つまりADDX, SUBDXのオペランドがBCDでなかった場合には、EITとはならないが、destやフラグに設定される結果は保証できない(インプリメント依存になる)ものとする。不正オペランド例外(IOE)としないのは、インプリメントの負担が大*

```
PACKHB src[.H],dest[.B]
```

```
RR=01,WW=00  src[04:07]==> dest[00:03],src[12:15]
```

```
==> dest[04:07]
```

```
PACKWH src[.W],dest[.H]  <<L2>>
```

```
RR=10,WW=01  src[04:07]==> dest[00:03],src[12:15]
```

```
==> dest[04:07]src[20:23]
```

```
==> dest[08:11],src[28:31]
```

```
==> dest[12:15]
```

```
PACKWB src[.W],dest[.B]
```

```
RR=10,WW=00  src[12:15]==> dest[00:03],src[28:31]
```

```
==> dest[04:07]
```

```
PACKLW src[.L],dest[.W]  <<LX>>
```

```
PACKLH src[.L],dest[.H]  <<LX>>
```

なお、PACKss, UNPKssにおいて、サイズの違いによってニモニックまで変えているのは、サイズの違いによって命令自体の意味もかなり変わると考えられるためである。つまり、一般の命令では、サイズの違いによってゼロ拡張や符号拡張を行なうだけであったが、PACKss, UNPKssでは命令のオペレーション自体がかなり異なっている。PACK, UNPKで、上

148

*きく、実行速度に影響が出るためである。

【0183】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・WW='11' のとき

・EaR がθ-SPのとき

・EaM が#imm__data.θSP+.θ-SPのとき

10 ・〈〈L1〉〉機能例外

・SUBDXの正しいビットパターンがデコードされたとき

〔ニモニック〕

PACKss src, dest (本発明装置ではサポートしない)

〔命令の機能〕

```
pack string into BCD
```

BCDへのパック

〔命令オプション〕なし

20 〔命令ビットパターンとアセンブラ表記〕図177に示す。

〔フラグ変化〕図178に示す。

〔解説〕srcをBCD(Binary Coded Decimal)にパックしてdestに転送する。実際には、ssにB, H, W, Lのいずれかの文字が入り、次のようなニモニックとオペレーションになる。

記の説明に含まれない不合理なサイズの組み合わせを指定した場合には、動作を保証しない(インプリメントに依存した値がdestに設定される)ものとする。アーキテクチャ上は予約命令例外(RIE)とするのが望ましいが、2つのオペランドのサイズの組み合わせによって予約命令例外(RIE)を検出するのはインプリメントの負担が大きいため、予約命令例外(RIE)とはし

ない。これは、異種サイズ間の論理演算の場合も同様である。srcのうち、destに影響を与えないフィールド(PACKHBの2⁷~2⁴のビットなど)については、0かどうかのチェックは行なわず、0でなくても無視する。文字コードをそのままパックするケースを考えると、0でないことの方が多い。

【0184】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・W='11' のとき
- ・EaR が@-SPのとき
- ・EaW がimm_data.@SP+ のとき

・〈〈L1〉〉機能例外

・PACKssの正しいビットパターンがデコードされたとき

〔ニモニック〕

UNPKss src, dest, adj (本発明装置ではサポートしない)

〔命令の機能〕

unpack BCD

BCDからのアンパック

〔命令オプション〕なし

図179に示す。

〔フラッグ変化〕図180に示す。

〔解説〕BCD(Binary Coded Decimal)のsrcのアンパックを行ない、アンパックした値に補正值adjを加えてdestに転送する。補正值adjを加えるのは、UNPK命令によって直接文字コードまで生成するためである。adjの加算は、BCDではなくバイナリの加算である。adjのサイズはdestのサイズと共通にWWフィールドによって指定される。実際には、ssにB, H, W, Lのいずれかの文字が入り、次のようなニモニックとオペレーションになる。図181に示す。PACK, UNPKで、上記の説明に含まれない不合理なサイズの組み合わせを指定した場合には、動作を保証しない(インプリメントに依存した値がdestに設定される)ものとする。アーキテクチャ上は予約命令例外(RIE)とするのが望ましいが、2つのオペランドのサイズの組み合わせによって予約命令例外(RIE)を検出するのはインプリメントの負担が大きいため、予約命令例外(RIE)とはしない。adjの加算によるオーバーフローは無視する。

【0185】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・W='11' のとき
- ・EaR が@-SPのとき
- ・EaW がimm_data.@SP+ のとき

・〈〈L1〉〉機能例外

・UNPKssの正しいビットパターンがデコードされたとき

【0186】12-10. スtring命令

「String」とは、8ビット、16ビット、32ビット、または64ビットのデータを任意の長さだけ連続して並べたデータタイプである。(SSCH命令に限り、連続ではなく、一定間隔で飛び飛びのアドレスにあるデータの集合もサポートしている。)

10 個々のデータの意味は特に決まっておらず、実際の文字コードになる場合、整数になる場合、浮動小数になる場合などがある。これは、ユーザ側で解釈する。Stringの範囲を示す方法には、

・Stringの長さ(データ数)を指定する方法

・String終了を示す文字(ターミネータ)を指定する方法

の2通りがあり、使用目的や言語によって適した方を選択する必要がある。本発明装置のString命令ではStringのデータ数がパラメータとなっているが、さらにオプションの終了条件という形でターミネータを与えることができ、両方の指定方法をサポートしている。本発明装置のString命令の特徴の一つとして、ポインタの増減量を自由に設定できるということがあげられる。そのため、Stringサーチ命令(SSCH命令)

20 を使ってテーブル検索や多次元配列のスキャンなども行なうことができる。また、本発明装置ではString命令SMOV, SCMP, SSCHの終了条件として大小比較や二値比較を含む豊富な条件が指定でき、これも大きな特徴となっている。String命令のうち、String

30 サーチ用のSSCH命令は、検索条件が終了条件として指定されるため、終了条件にのみ意味がある命令となっている。本発明装置のString命令で指定できる条件(eeee)については、付録を参照のこと。本発明装置では〈〈L2〉〉仕様となっている、終了条件OUTU-ZEはサポートしない。String命令の用途としては、文字どおり8/16ビットの文字列を処理するもののほか、特定のビットパターンのサーチ、メモリのブロック転送、構造体の代入、メモリ領域のクリアなどの応用がある。String命令は任意長ビットフィールド命令と同じく不定長のデータを扱うため、実行中の割り込み受付、実行再開の機能が不可欠である。一方、String命令自体がコンパイラの生成するコードとなることはほとんどなく、アセンブラで書かれたサブルーチンとして提供されることが多い。そのため、対称性やアドレッシングモードについての制限はあまり問題にならない。したがって、本発明装置のString命令では、オペランドや実行途中の状態保持のために固定番号のレジスタ(R0-R4)を使うようになっている。主なレジスタの使い方は次のようになる。

151

152

R0: ソース側ストリングの先頭アドレス
 R1: デスティネーション側ストリングの先頭アドレス
 R2: スtringの長さ、データ数
 R3: 終了条件の比較値(1)
 R4: 終了条件の比較値(2)

このうち、ストリングの長さを表わすR2はエレメント数であって、バイト数ではない。R2は符号なしの数として扱われ、R2=0の場合はエレメント数による命令終了は行なわないという意味に解釈される。つまり、エレメント数による終了を避けたい場合には、R2=0として命令を実行すれば良いことになる。インプリメント上は、ストリング命令の実行パターンは次のようになるのが一般的である。

```
do {
    ...
    R2 - 1 ==> R2;
    check_interrupt;
} while (R2 !=0);
```

ただし、R2=0とした場合に、エレメント数をH' 10000000と考えるか、無限大(エレメント数の*

V_flag エレメント数(ストリング長)による終了

F_flag 終了条件(eeee)による終了

この時、複数の終了条件を区別するために M_flagを使用する。

M_flagの変化については付録参照。

これ以外に終了要因のないSCMP, SSCHでは、V_flagとF_flagの変化が相補的となる。SCMPの場合には、これ以外に比較データの不一致によって命令を終了する場合がある。

〔ニモニック〕

/F アドレス増加の方向に処理する

/B アドレス減少の方向に処理する

/ 終了条件各種(eeee) (本発明装置では〈L2〉となっているOUTU-ZEはサポートしない。)

〔命令ビットパターンとアセンブラ表記〕図182に示す。

〔フラッグ変化〕図183に示す。

〔解説〕ストリングの転送を行なう。減少方向の操作をするストリング命令SMOV/Bでは、最初にR0, R1で指定するアドレスが、操作対象となるストリングの占めるアドレスの最大アドレス+1を指し、R0, R1をプリデクリメントしながら操作を進める。SMOVでsrcとdestがオーバーラップしていた時に、/F, /Bのうち正しくない方のオプションを指定した場

*チェックを行なわない) と考えるかはインプリメントに依存する。つまり、H' 100000000回のエレメント操作を行なっても命令を終了しなかった場合に、その後の動作はインプリメント依存になるものとする。ただし、エレメント数以外の要因により命令を終了した場合(R2=0の場合は普通こうなる)には、命令終了後のR2の値(付録11参照)が正しくセットされていなければならない。実際には、SSCH/RでR5=0を指定したような特殊な場合を除いて、H' 10000000000回のエレメント操作を行なう間にアドレス変換例外(ATRE)やバスアクセス例外(BAE)を起こし、命令中断となるのが普通である。ストリング命令はいろいろな要因で終了するため、それらを区別するためにフラッグを用いる。それぞれのフラッグの意味は、次のようになっている。

※SMOV

〔命令の機能〕

copy string

ストリングのコピー

〔命令オプション〕

合の動作は、BVCPY, BVMAPと同じように保証しないものとする。つまり、インプリメントや命令実行中断の有無によって異なる場合がある。これは、高機能命令の特徴を生かしてパイプライン的なメモリアクセスをした場合に、メモリアクセスの順番が変わることがあり、必ずしも前エレメントの書き込みが終わってから次エレメントの読み出しを行なうとは限らないからである。なお、逆方向の処理オプション/Bは、この命令SMOV/Bに限り〈L2〉ではなく、〈L1〉となっている。任意長ビットフィールド命令やストリン

153

154

グ命令などの高機能命令の詳細仕様や、命令終了後のレジスタ値については、付録11を参照のこと。

*【0187】〔プログラム例外〕

・予約命令例外

- ・SS='11' のとき
- ・P='1' のとき
- ・Q='1' のとき
- ・eeee='0111' ~ '1111' のとき

〔ニモニック〕

※compare string

SCMP

10 スtringの比較

〔命令の機能〕

※〔命令オプション〕

/F アドレス増加の方向に処理する

/B アドレス減少の方向に処理する

<<L2>> (本発明装置ではサポートする)

/ 終了条件各種(eeee) (本発明装置では <<L2>> となっているOUTU-ZB はサポートしない。)

〔命令ビットパターンとアセンブラ表記〕図184に示す。

〔フラッグ変化〕図185に示す。

〔解説〕String src1, src2, の比較を行なう。2つのStringが一致している間は比較を続け、一致しない文字が見付かれれば比較を終了する。SCMP命令では、CMP命令と同様に、src2-src1の結果をもとにしてフラッグの設定を行なう。例えばL_flagは、src1に対してsrc2の方がより小さ★

1. エレメント (データ) 数(R2)による終了

V_flag = 1

2. 終了条件による終了

F_flag = 1, 終了要因によって M_flagが変化

3. 比較中のデータの不一致による終了

Z_flag = 0, 比較結果によって L_flag, X_flagが変化

L_flagは、最後のデータを符号付きと見て比較した時の比較結果

果

X_flagは、最後のデータを符号なしと見て比較した時の比較結果

果

1~3の要因のうち、2と3のチェックを同時に行なうことは可能であるが、1の要因のチェックは、2, 3とは別のフェーズで行なわれる。したがって、2と3が同時に成立することはない。1と2や1と3が同時に成立することはない。1, 2, 3の少なくとも一つの終了要因が成立した場合にSCMP命令が終了する。比較するデータが一致している間は、その値(src1=src2)が終了条件のテスト対象となるが、データに不一致があった場合には、R0により示されるsrc1の方を終了条件のテスト対象とする。ただし、不一致によりSCMP命令が終了した場合には、終了条件が成立した

★いということを示す。src1-src2の結果をもとにしてフラッグの設定を行なうのではない。終了条件を持つSCMP命令の応用としては、テキストを一行単位で比較する(R3に改行の文字コードをセットしてSCMP/EQを実行)、数字が続く間だけ比較する、全角文字が続く間だけ比較する、といったものがある。SCMPでは、命令を終了させる要因が次の3つ存在し、フラッグ変化によってそれらを見分けることができる。

40 かどうかという情報は必要ないことが多いため、これは単なる約束に過ぎない。また、終了条件が満たされないと意味を持たないM_flagについては、別の終了要因により終了した場合に、意味が不明確となる。そのような場合のフラッグ変化は、0になるものと決めておく。Z_flag, L_flag, X_flagについては、一致、不一致にかかわらず必ず最後のデータの比較結果を反映する。したがって、3. 以外の要因で終了した場合(データが一致している場合)には、自動的にZ_flag=1, L_flag=0, X_flag=0となる。なお、SCMPでは符号なしデータ、符号付きデータの両方を扱うため、L_flagにエレメント

155

を符号付きデータと考えた時の比較結果が入り、X__f
lagにエレメントを符号なしデータと考えた時の比較
結果が入るようになっている。BTRONの文字コード
は符号なしで扱う必要があるし、一般の整数を扱うので*

+ --- フラッグ本来の意味ではなく、約束としてこうなっていること

を示すものである

*A --- src1=src2 が終了条件のどれに該当するか、による

*B --- src1が終了条件のどれに該当するか、による

*C --- src1 < src2 かsrc2 < src1 か、による

/Bオプションは〈L2〉であるが本発明装置では
サポートする。

【0188】〔プログラム例外〕

・予約命令例外

・SS='11' のとき

・P='1' のとき

・Q='1' のとき

・eeee='0111' ~ '1111' のとき

〔ニモニック〕

SSCH

20

※

/ 終了条件各種(eeee) (本発明装置では〈L2〉となっているOUTU-ZE はサ
ポートしない。)

〔命令ビットパターンとアセンブラ表記〕図187に示
す。

〔フラッグ変化〕図188に示す。

〔解説〕ストリングをサーチし、条件に合うエレメント
を見つけて出す。'/R' オプションの場合には、R5の
正負にかかわらず、常にエレメントの比較後にR0が更
新(ポストインクリメントまたはポストデクリメント)★

・予約命令例外

・SS='11' のとき

・P='1' のとき

・eeee='0111' ~ '1111' のとき

〔ニモニック〕

SSTR

〔命令の機能〕

store characters

同一データを繰り返し書き込み(ストリングのフィル)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図189に示
す。

〔フラッグ変化〕図190に示す。

〔解説〕先頭アドレス(R1)と長さ(R2)により指
定されたメモリ領域に、R3の値を繰り返し書き込む。
SSTR命令では終了条件が意味を持たないため、終了
条件の指定は行なわない。なお、ストリング命令でR2
=0の場合はエレメント数による終了を行なわないが、

156

*あれば、符号付きのデータを扱う必要もある。SCMP
のフラッグ変化をまとめると図186のようになる。○
は終了要因が満たされたことを示し、×は終了要因が満
たされないことを示す。

※〔命令の機能〕

find a character in a str
ing

ストリングのサーチ

〔命令オプション〕

/F アドレス増加の方向に処理する

(ポインタ値をエレメントサイズずつ増加させる)

/R ポインタの増加値はR5で指定される

★される。SSCH/RのR5のサイズはR0のポインタ
サイズと等しくなる。つまり、本発明装置32では32
ビット固定であり、本発明装置64ではPビットまたは
モードによって指定される。SS(R3, R4, エレメ
ントサイズ)とは独立である。

【0189】〔プログラム例外〕

40

SSTR命令の場合はエレメント数による終了が唯一の
終了要因であり、R2=0を指定すると無限ループを形
成することになる。これについては、ハードウェアでは
特に対処せず、プログラム側で注意してもらうことにす
る。ただ、実行中の割り込み受付や再実行は可能な
で、この命令によって間違って無限ループに入ったとし
ても、タスクやプロセスのスケジューリング等には影響
しない。通常は複数の命令によって構成される無限ル
ープが、たまたま一命令にまとめられただけと考える。R
2=0を不正オペランド例外IOEとしないのは、他の
ストリング命令との仕様の統一や、インプリメントの負
担、高速化を考えたためである。このほか、SSCHや
QSCH命令でも、パラメータや終了条件の指定によっ
ては一つの命令で無限ループを形成する場合がある。

157

【0190】〔プログラム例外〕

・予約命令例外

・SS='11' のとき

・P='1' のとき

【0191】12-11. キュー命令

本発明装置ではキューを操作するための命令として、QINS (エントリの挿入)、QDEL (エントリの削除)、QSCH (エントリのサーチ) の3つの命令が用意されている。本発明装置でサポートしているキューは、各キューエントリの先頭から1番目と2番目のデータが絶対アドレスのリンクポインタとなった、ダブルリンクのキューである。キューエントリの先頭にあるデータが次のキューエントリへのポインタとなり、キューエントリの2番目にあるデータが前のキューエントリへのポインタとなっている。キュー命令の仕様は、キューヘッダをキュー命令のオペランドとして直接指定できるように、以下のような方針で決められている。

1. QDELでは、指定したエントリではなく直後のエントリが削除される。QUEUE HEADをオペランドとして指定した場合には、先頭のエントリが削除されることになる。QSCH/Bで見付けたエントリを削除する場合や、キューの最後のエントリを削除する場合には間接参照が必要になるが、QSCH/Fで見付けたエントリを削除する場合やキューの先頭のエントリを削除する場合に比較すれば、頻度は少ないと考えられる。

2. QINSでは、指定したエントリの直前に新しいエントリが挿入される。QUEUE HEADをオペランドとして指定した場合には、キューの最後にエントリが挿入されることになる。これについては2つの考え方がある。QDEL命令との対称性を考えると、QINSでは指定したエントリ (あるいはQUEUE HEAD) の直後にエントリを挿入する方が望ましい。これは、QINSで挿入したエントリをQDELで削除するために、同じオペランドを指定できるからである。また、キューをスタック (LIFO) 的な使い方にする場合にも、このような仕様の方が良い。一方、キューをFIFOで使う場合には、QINSでキューの最後にエントリを挿入し、QDELではキューの先頭のエントリを削除することが多い。こちらの方がキュー本来の使い方であるし、ITRONでもそのような例がある。したがって、後者の仕様にする。

3. QSCHでは、現在のエントリではなく直後のエントリからサーチを始める。QUEUE HEADをオペランドとして指定した場合には、キューの先頭からサーチを始めることになる。また、サーチが成功した場合に次のサーチを行なうには、そのままう一度QSCHを実行すればよい。この考え方は、他の高機能命令 (ストリング、任意長ビットフィールド操作) とは異なったものである。すなわち、ストリング命令では現在ポインタ

158

の指しているデータ自体からサーチが始まり、連続サーチを行なう場合には別命令でポインタを更新しておく必要がある。これはキュー命令とは異なった動きである。しかし、キューの場合にはヘッダが別になっているという事情があり、扱いが異なっているため、別の仕様にしても構わないと判断した。

4. 空のキューをフラッグで判定する。QINSで空のキューにデータを挿入した場合、QDELでエントリの削除の結果キューが空になった場合には、Z__flagをセットする。また、QDELで空のキューからエントリを削除しようとした場合は、エラーなのでポインタの付け変えは行なわないが、この時V__flagをセットする。

〔ニモニック〕

QINS entry, queue

〔命令の機能〕

insert a new entry into a queue

ダブルリンクのキューへ挿入

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図191に示す。

〔フラッグ変化〕図192に示す。

〔解説〕entryで指定される新しいキューのエントリが、queueで示されるキューエントリの直前に挿入される。queueで指定されるキューエントリがキューヘッダであった場合には、この命令によって、キューの最後に新しいエントリが挿入されることになる。命令実行前にキューが空であったかどうかによって、Z__flagがセットされる。

〔32ビットで処理を行なう場合のQINS命令のオペレーション〕図193に示す。

〔実行前〕図194に示す。

〔実行後〕図195に示す。EaMqP, EaMaP2で指定されるアドレッシングモードでは、レジスタ直接Rn, @-SP, @SP+, #imm_dataのモードは使用できない。なお、QINSでは、命令の実行のために直接必要ではない部分のデータ構造のチェック (queueの前後のキューエントリのリンク関係など) は特に行なわない。オペレーションに書かれている通りの動作を行なう。

【0192】〔プログラム例外〕

・予約命令例外

・+= '0' のとき

・-= '1' のとき

・EaMqP がRn, #imm_data, @SP+, @-SPのとき

・BaMqP2がRn, #imm_data, @SP+, @-SPのとき

〔ニモニック〕

QDEL queue, dest

〔命令の機能〕

remove an entry from a queue

ダブルリンクのキューエントリを削除

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図196に示す。

〔フラッグ変化〕図197に示す。

〔解説〕queueで指定されたキューエントリの次のエントリを削除し、削除されたエントリのアドレスをdestにセットする。削除されたエントリのアドレスをdestにセットするのは、それ以後削除したエントリを操作することが多いためである。queueとしてキューヘッダを指定した場合には、キューの先頭のエントリが削除されることになる。queueで指定されたキューが空のキューであった場合は、命令の実行ができない。この時、EITは起動せず、Vflag, Zflagのセットだけを行なって命令を終了する。dest*

・予約命令例外

・t='0' のとき

・W='1' のとき

・EaRqP がRn, #imm_data, @SP+, @-SPのとき

・BaW!S が#imm_data, @SP+, @-SPのとき

〔ニモニック〕

QSCH

〔命令の機能〕

/NM R6 のマスクなし

/MR R6 のマスクあり <<L2>> (本発明装置ではサポートしない)

/F キューの順方向にサーチ

/B キューの逆方向にサーチ <<L2>> (本発明装置ではサポートする)

/ 終了条件各種(eeee) (本発明装置では<<L2>>となっているOUTU-ZE はサポートしない)

〔命令ビットパターンとアセンブラ表記〕図201に示す。セットの必要なのは、R0, R2, R3 (オプション), R4 (オプション), R5, R6 (オプション) であり、結果が入るのは、R0, R1である。続けて次のサーチを行なうことができる。

〔フラッグ変化〕図202に示す。

〔解説〕キューのエントリをサーチし、条件に合ったものを見付ける。逆方向のサーチ機能/B、およびマスクの機能/MRは<<L2>>となっている。本発明装置では逆方向のサーチ機能/Bをサポートする。マスクの機能/MRはサポートしない。この命令はキューの長さ依存した処理量となるので、ストリング命令と同じように実行中の中断に対する考慮が必要である。したがって、オペランドと途中の実行状態は固定番号のレジスタ

*tは無変化となる。dest/EaW!Sでは、@-SPのモードを禁止している。これは、キューが空でVflagがセットされ、destの転送ができない場合に、destに@-SPが指定されていると命令動作がまぎらわしくなるためである。

〔32ビットで処理を行なう場合のQDEL命令のオペレーション〕図198に示す。

〔実行前〕図199に示す。

〔実行後〕図200に示す。EaRqPで指定されるアドレッシングモードでは、レジスタ直接、@-SP, @SP+, #imm_dataのモードは使用できない。なお、QDELでは、空のキューの判定以外のチェック、命令の実行のために直接必要ではない部分のデータ構造のチェック(queueの前後のキューエントリのリンク関係など)は特に行なわない。オペレーションに書かれている通りの動作を行なう。

【0193】〔プログラム例外〕

※search queue entries

キューのサーチ

※〔命令オプション〕

に置く。サーチ条件としては、マスク(特定ビットの抽出)と比較が用意されている。マスクはフラッグのサーチに用い、比較は優先度の処理などに用いる。比較条件の指定は、ストリング命令の終了条件の指定と同じである。キューの終りを判定するために、キューのエントリアドレスとキューの終了アドレスR2との比較を行ない、一致した場合には命令を終了する。R2との比較によって命令を終了した場合、すなわち、それまでにサーチ条件を満たすものがなく、サーチ失敗であった場合には、Vflagをセットして命令を終了する。EITは起動しない。QSCH命令の条件指定によっては、一つの命令の中で無限ループに入ることがある。これについては、ハードウェアでは特に対処せず、プログラム側で注意してもらうことにする。ただ、実行中の割り込み

161

受付や再実行は可能なので、ユーザプログラムの中で間違っ
て無限ループに入ったとしても、タスクやプロセスのスケ
ジューリング等には影響しない。通常は複数の命令によっ
て構成される無限ループが、たまたま一命令にまとめられ
ただけと考える。サーチが終了した時に、R0は指定した条
件に合うエントリを、R1はその直前のエントリを指してい
る。R1は、シングルリンクのキューの時にエントリを削除
するために使用することができる。また、QDELでは指定
したエントリの次のエントリが削除されるので、QSCH/F
で見付けたエントリ自体を削除する場合には、QSCH実行
後、@R0ではなく@R1をパラメータとしてQDELを実行す
ればよい。一般に、R0、R2にQUEUE HEADのアドレスを
セットしてQSCH命令を実行することにより、キューが空の
場合を含めてキュー全体のサーチを行なうことができる。
QSCHは、シングルリンクキューとダブルリンクキューで共
用することを狙った命令である。

【QSCHのオペレーション】図203に示す。このうち、
check_interruptは、外部から割り込みがかかっているか
どうかを調べ、割り込みがかかっていたら、QSCHの実行
を中断して割り込み処理を始めるというものである。割り
込み処理終了後にQSCH命令の残りの部分を実行する。

【実行前】図204に示す。

【実行後】図205に示す。

【0194】〔プログラム例外〕

・予約命令例外

・SS='11' のとき

・eeee='0111' ~ '1111' のとき

・m='1' のとき

【0195】12-12. ジャンプ命令

〔ニモニク〕

BRA newpc

〔命令の機能〕

branch always

ジャンプ (PC相対)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図206に示す。

〔フラグ変化〕図207に示す。

〔解説〕BRA命令は、PC相対のみのアドレッシングをサ
ポートするジャンプ命令である。ディスプレースメントの
サイズとして、BRA:Dでは8ビットが、BRA:Gでは8ビ
ット、16ビット、32ビット、64ビットが利用できる。本
発明装置の命令は必ず偶数アドレスから始まるので、短
縮形のBRA:D命令では、#d8を2倍して使用する。すな
わち、

PC + #d8 * 2 ==> PC

162

となる。BRA:GでSS=00を指定した場合には、#dSを2
倍せずにそのまま使用する。BRA:Gでnewpcが16ビ
ットの場合、JMP@(#dS:16, PC)と命令機能、コード
サイズともに同じであるが、実行サイクル数を短くでき
る可能性があるため、別命令となっている。BRA:Gで、
newpcが奇数であった場合には、ジャンプ先が奇数アド
レスになるため、奇数アドレスジャンプ例外(OAJE)と
なる。これは、Bcc:G, BSR:G, JMP, JSR命令も同様
である。BRA:D, Bcc:D, BSR:Dでは、オペランドを2
倍して使用するため、OAJEは発生しない。BRA:G, Bcc:
G, BSR:GでSS=00の場合、オペランドサイズは8ビ
ットであるが、#dSフィールドは16ビットとなる。この
時、#dSフィールドは下位8ビットのみを使用し、上位8
ビットには必ず0を入れておかなければならない。上位8
ビットが0でない場合は、これによって表現されるデー
タがインプリメント依存の不定値になるものとする。つ
まり、BRA:G命令の場合は、ジャンプ先が不定となる。
EITにはしない。本発明装置ではこの命令に対し、動
的ブランチ予測処理をする。

【0196】〔プログラム例外〕

・予約命令例外

・SS='11' のとき

・P='1' のとき

・奇数アドレスジャンプ例外

・奇数アドレスにジャンプしたとき

30 〔ニモニク〕

Bcc newpc

〔命令の機能〕

branch conditionally

条件ジャンプ (PC相対)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図208に示す。

〔フラグ変化〕図209に示す。

〔解説〕Bcc命令は、PC相対のみのアドレッシングをサ
ポートする条件ジャンプ命令である。ディスプレースメン
トのサイズとして、Bcc:Dでは8ビットが、Bcc:Gでは8
ビット、16ビット、32ビット、64ビットが利用できる。本
発明装置の命令は必ず偶数アドレスから始まるので、Bcc:
D命令では、#d8を2倍して使用する。すなわち、

if(cccc)

PC + #d8 * 2 ==> PC

となる。Bcc:GでSS=00を指定した場合には、

50 #dSを2倍せずにそのまま使用する。Bccの条件指

163

定部分 ('cc' 部分)の詳細とニモニック、ccccのビットパターンについては、付録を参照のこと。Bccで未定義の条件を指定した場合には、予約命令例外(RIE)となる。Bcc:Gで条件不一致のためジャンプしなかった時は、本発明装置では奇数アドレスジャンプ例外(OAJE)を発生する場合と、発生しない場合がある。本発明装置ではこの命令に対し、動的ブランチ予測処理をする。

【0197】〔プログラム例外〕

・予約命令例外

・SS='11' のとき

・P='1' のとき

・cccc='1110' ~ '1111' のとき

・奇数アドレスジャンプ例外

・奇数アドレスにジャンプしたとき

〔ニモニック〕

BSR newpc

〔命令の機能〕

jump to subroutine

サブルーチンジャンプ(PC相対)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図210に示す。

〔フラッグ変化〕図211に示す。

〔解説〕BSR命令は、PC相対のみのアドレッシングをサポートするサブルーチンジャンプ命令である。PCの値がスタックに退避される。ディスプレースメントのサイズとして、BSR:Dでは8ビットが、BSR:Gでは8ビット、16ビット、32ビット、64ビットが利用できる。本発明装置の命令は必ず偶数アドレスから始まるので、BSR:D命令では、#d8を2倍して使用する。すなわち、

$PC + \#d8 * 2 \Rightarrow PC$

となる。BSR:GでSS=00を指定した場合には、#dSを2倍せずにそのまま使用する。BSR, JSR命令でスタックに退避されるPC値としては、その次の命令の先頭アドレスを使用する。これに対して、実効アドレスの計算のためにPCを参照する場合(BSRなどで暗黙にPCを参照する場合を含む)には、その命令(次の命令ではない)の先頭アドレスをPC値として使用するので、注意が必要である。BSR, JSRでは旧のPCがスタックにセーブされるが、SPのアラインメントに関しては特にチェックしない。SPが4の倍数でない場合にも、そのまま実行される。本発明装置ではこの命令に対し、動的ブランチ予測処理をする。

【0198】〔プログラム例外〕

164

・予約命令例外

・SS='11' のとき

・P='1' のとき

・Q='1' のとき

・奇数アドレスジャンプ例外

・奇数アドレスにジャンプしたとき

〔ニモニック〕

10 JMP newpc

〔命令の機能〕

address of src \Rightarrow PC

ジャンプ

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図212に示す。

〔フラッグ変化〕図213に示す。

〔解説〕newpcの実効アドレスにジャンプする。一般のアドレッシングモードが使用可能なジャンプ命令である。case文の実行などにおいては、ジャンプテーブルを参照してジャンプ先アドレスを決める場合がある。これはJMP命令と付加モードによるインデックスアドレッシングとを組み合わせるにより実現する。

【0199】〔プログラム例外〕

・予約命令例外

・BaA がRn, #imm_data, @SP+, @-SPのとき

・奇数アドレスジャンプ例外

・奇数アドレスにジャンプしたとき

〔ニモニック〕

30 JSR newpc

〔命令の機能〕

jump to subroutine

サブルーチンジャンプ

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図214に示す。

〔フラッグ変化〕図215に示す。

〔解説〕newpcの実効アドレスにサブルーチンジャンプする。PCの値がスタックに退避される。BSR, JSR命令でスタックに退避されるPC値としては、その次の命令の先頭アドレスを使用する。これに対して、実効アドレスの計算のためにPCを参照する場合(BSRなどで暗黙にPCを参照する場合を含む)には、その命令(次の命令ではない)の先頭アドレスをPC値として使用するので、注意が必要である。

【0200】〔プログラム例外〕

165

・予約命令例外

・P='1' のとき

・EaR がRn, #imm_data, @SP+, @-SPのとき

・奇数アドレスジャンプ例外

・奇数アドレスにジャンプしたとき

〔ニモニック〕

ACB step, xreg, limit, newpc

〔命令の機能〕

add, compare and branch

インデクス値を増加するループ命令

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図216に示す。

〔フラッグ変化〕図217に示す。

〔解説〕加算、比較、条件ジャンプを一命令にした複合命令であり、ループのプリミティブとして利用する。step, xreg, limitは符号付き整数として演算、比較される。stepは必ず正の値でないと条件ジャンプの意味がないが(xregが終了値と反対の方向に変化することになる)、stepの正負のチェックは*

xreg + step ==> xreg

/* オーバーフローの場合は下位ビットのみ有効*/

if (xreg < limit) then PC + #dS8 ==>PC endif

newpcが奇数であった場合には、ジャンプ先が奇数アドレスになるため、奇数アドレスジャンプ例外(OAJE)となる。本発明装置では、終了条件満足のためジャンプしなかった時も、奇数アドレスジャンプ例外(OAJE)を発生する。

〔詳細仕様調整中〕ACB, SCB命令でSS≠00の時には、#dS8のフィールドは使用しない。この時、もし#dS8のフィールドが0になっていなくても、単に無視される。ただし、マニュアル上は、#dS8のフィールドには常に0を入れるようにしておく。本発明装置ではこの命令に対して、動的ブランチ予測処理をする。

【0201】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・XX='11' のとき

・SS='11' のとき

・P='1' のとき

・EaR が@-SPのとき

・EaRXが@-SPのとき

・奇数アドレスジャンプ例外

・奇数アドレスにジャンプしたとき

166

*行なわず、オペレーションに書かれている通りの動作をそのまま行なう。ACB命令では、ループ命令として高速実行ができるように、step加算時のオーバーフローのチェックは行なわない。stepを加算した結果オーバーフローが起こり、符号が反転した場合には、符号反転した正しくない値がそのままlimitと比較される。ただし、比較のためのlimit-xregの減算がオーバーフローしたとしても、xreg<limitの比較は正確に行なわれる。ACB, SCBではPC相対でジャンプを行なう。SS=00でディスプレースメントが8ビットになる場合も、SS≠00の場合と同様に、#dS8は2倍せずにそのまま使用する。SS≠00の場合は、#dS8のフィールドは使用せず(0にする)、SSで指定されたサイズ(16, 32, 64ビット)のデータが#dS8の直後に続く。例えば、

ACB:Q #1.R0.#4.label

で、labelとACB:Q命令のアドレスの差がH'1234であった場合は、図218に示すビットパターンになる。これは、固定長ビットフィールド命令の:IFOフォーマットでも同じである。

〔ACBのオペレーション〕

〔ニモニック〕

SCB step, xreg, limit, newpc

〔命令の機能〕

subtract, compare and branch

インデクス値を減少するループ命令

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図219に示す。

〔フラッグ変化〕図220に示す。

〔解説〕減算、比較、条件ジャンプを一命令にした複合命令であり、ループのプリミティブとして利用する。step, xreg, limitなどは符号付き整数として比較される。stepは必ず正の値でないと条件ジャンプの意味がないが、(xregが終了値と反対の方向に変化することになる)。stepの正負のチェックは行なわず、オペレーションに書かれている通りの動作をそのまま行なう。SCB命令では、ループ命令として高速実行ができるように、step減算時のオーバーフローのチェックは行なわない。stepを減算した結果オーバーフローが起こり、符号が反転した場合には、符号反転した正しくない値がそのままlimitと比較される。ただし、比較のためのlimit-xregの減算

167

がオーバーフローしたとしても、 $xreg < limit$ の比較は正確に行なわれる。ACB, SCBではPC相対でジャンプを行なう。SS=00でディスプレースメントが8ビットになる場合も、SS≠00の場合と同様に、#dS8は2倍せずにそのまま使用する。SS≠0*

$xreg - step ==> xreg$

/*オーバーフローの場合は下位ビットのみ有効*/

if ($xreg \geq limit$) then PC + #dS8 ==>PC endif

newpcが奇数であった場合には、ジャンプ先が奇数アドレスになるため、奇数アドレスジャンプ例外(OAJE)となる。本発明装置では、終了条件満足のためジャンプしなかった時も、奇数アドレスジャンプ例外(OAJE)を発生する。

〔詳細仕様調整中〕ACB, SCB命令でSS≠00の時には、#dS8のフィールドは使用しない。この時、もし#dS8のフィールドが0になっていなくても、単に無視される。ただし、マニュアル上は、#dS8のフィールドには常に0を入れるようにしておく。本発明装置ではこの命令に対して、動的ブランチ予測処理をする。

【0202】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・XX='11' のとき
- ・SS='11' のとき
- ・P='1' のとき
- ・EaR が@-SPのとき
- ・EaRXが@-SPのとき

- ・奇数アドレスジャンプ例外
- ・奇数アドレスにジャンプしたとき

〔ニモニック〕

ENTER local, reglist

〔命令の機能〕

create new stack frame

スタックフレームの形成、高級言語用サブルーチンジャンプ

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図221に示す。

〔フラッグ変化〕図222に示す。

〔解説〕高級言語用のスタックフレームを形成する。ENTERのlocalは符号付きとして扱われ、localのサイズが小さい場合には、localの値が符号拡張される。localが負の場合は意味がないスタック

168

*0の場合は、#dS8のフィールドは使用せず(0にする)、SSで指定されたサイズ(16, 32, 64ビット)のデータが#dS8の直後に続く。

[SCBのオペレーション]

クフレームが形成されるが、特にチェックは行なわず、オペレーションに書かれている通りの動作を行なう。この点は、ACB, SCBのstepと同じである。
オペレーション:

FP -> ↓TOS

SP -> FP

SP - local -> SP

registers(mask) -> ↓TOS

高級言語用のスタックフレームの詳細は、付録を参照のこと。退避するレジスタのビットマップ指定LnXLは、図223のように行なう。LnXLは、EaRの拡張部よりも後に置かれる。ENTERのreglistでbit0, bit1(SP, FP)を指定した場合には、単にその指定が無視されるものとする。bit0, bit1が"1"であっても、SP, FPは転送されない。これを不正オペランド例外(IOE)としないのは、インプリメントの負担が大きく、実効速度に影響が出るためである。ただし、チェックが行なわれていなくても、FP, SPのビットには必ず"0"を入れてもらうように、マニュアル等で指導する必要がある。FP, SPのアラインメントに関しては特にチェックしない。FP, SPが4の倍数でない場合にも、オペレーションに書かれた通りの実行が行なわれる。ENTER:Gのlocalオペランドがメモリ上にあり、それがENTER命令の実行に伴って形成されるスタックフレーム領域と重なっていた場合には、命令再実行がきわめて難しくなる。そこで、ENTER:G, JRNG:G、および対称性からEXITD:G命令では、メモリアクセスを伴うアドレッシングモード、つまりレジスタ直接Rnとイミディエート以外のアドレッシングモードは、すべて禁止している。この命令のオペランドとして動的な値を設定したい場合には、テンポラリレジスタを一つ用意し、レジスタ直接Rnのモードを利用するということになる。localとしてFP, SPを指定した場合の動作は、インプリメント依存である。

【0203】〔プログラム例外〕

169

・予約命令例外

・X='1' のとき

・+= '0' のとき

・--='1' のとき

・P='1' のとき

SS='11' のとき

・EaR!M が#imm_data, Rn 以外のとき

〔ニモニック〕

EXITD reglist, adjsp

〔命令の機能〕

exit and deallocate parameters

高級言語用サブルーチンリターンとパラメータ解放

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図224に示す。

〔フラグ変化〕図225に示す。

〔解説〕高級言語用のスタックフレーム解放とレジスタの復帰を行ない、サブルーチンから戻る。その後 adjsp を SP に加え、スタック上に残っていたサブルーチンのパラメータを捨てる。EXITD の adjsp は符号付きとして扱われ、adjsp のサイズが小さい場合には、adjsp の値が符号拡張される。adjsp が負の場合は意味のない動作をするが、特にチェックは行なわず、オペレーションに書かれている通りに実行を行なう。この点は、ACB, SCB の step と同じである。

オペレーション:

adjsp ==> tmp

↑TOS ==> registers(mask)

FP ==> SP

↑TOS ==> FP

↑TOS ==> PC

sp + adjsp ==> SP

高級言語用のスタックフレームの詳細は、付録を参照のこと。復帰するレジスタのビットマップ指定 LxXL は、図226のように行なう。LxXL は、EaR の拡張部よりも後に置かれる。EXITD の reglist で bit14, bit15 (SP, FP) を指定した場合には、単にその指定が無視されるものとする。bit14, bit15 が "1" であっても、SP, FP は転送されない。これを不正オペランド例外 (IOE) としないのは、インプリメントの負担が大きく、実効速度に影響が出るためである。ただし、チェックが行なわれていなくても、FP, SP のビットには必ず '0' を入れてもらうように、マニュアル等で指導する必要がある。

170

10 FP, SP のアラインメントに関しては特にチェックしない。FP, SP が 4 の倍数でない場合にも、オペレーションに書かれた通りの実行が行なわれる。EXITD で、スタックから復帰されたリターンアドレスが奇数であった場合には、ジャンプ先が奇数アドレスになるため、奇数アドレスジャンプ例外 (OAJE) となる。EXITD のオペランド adjsp/EaR!M では、メモリアクセスを伴うアドレッシングモード、つまりレジスタ直接 Rn とイミディエート以外のアドレッシングモードは、すべて禁止している。この命令のオペランドとして動的な値を設定したい場合には、テンポラリレジスタを一つ用意し、レジスタ直接 Rn のモードを利用するということになる。adjsp にレジスタ直接 Rn のモードを利用し、reglist に同じレジスタ Rn が含まれていた場合には、adjsp として、レジスタ復帰前の値を使用する。つまり、スタック中に退避されていた EXITD 命令実行後のレジスタ値ではなく、EXITD 命令実行前のレジスタ値が adjsp となる。adjsp として FP, SP を指定した場合の動作は、インプリメント依存である。

30 【0204】〔プログラム例外〕

・予約命令例外

・X='1' のとき

・+= '0' のとき

・--='1' のとき

・P='1' のとき

SS='11' のとき

・EaR!M が#imm_data, Rn 以外のとき

40 〔ニモニック〕

RTS

〔命令の機能〕

return from subroutine

サブルーチンからのリターン

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図227に示す。

〔フラグ変化〕図228に示す。

〔解説〕サブルーチンからのリターンを行なう。

171
オペレーション：
↑TOS → PC

RTSで、スタックから復帰されたリターンアドレスが奇数であった場合には、ジャンプ先が奇数アドレスになるため、奇数アドレスジャンプ例外(OAJE)となる。

【0205】〔プログラム例外〕

・予約命令例外

・P='1' のとき

・奇数アドレスジャンプ例外

・リターンアドレスが奇数であったとき

〔ニモニック〕

NOP

〔命令の機能〕

no operation

ノーオペレーション

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図229に示す。

〔フラッグ変化〕図230に示す。

〔解説〕何もしない。

【0206】〔プログラム例外〕

・予約命令例外

・P='1' のとき

〔ニモニック〕

PIB

〔命令の機能〕

purge instruction buffer

命令キャッシュやパイプラインの整合性をとる

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図231に示す。

〔フラッグ変化〕図232に示す。

〔解説〕命令パイプライン、命令キュー、命令キャッシュなど、命令実行の高速化のためのバッファ類をすべてページし、メモリ上に置かれた命令列とプロセッサの内部状態との整合性を保証する。この命令は、これから実行すべき命令コードが、以前（リセット時あるいは前回のPIB命令実行時）から変更されている可能性があるということを、プロセッサに通知するために使用する。本発明装置では、パイプラインや命令キュー、命令キャッシュの制御を簡単化するため、プログラムにより命令コードを書き換えることは許されていない。つまり、自分自信で書き換えを行なった命令コードをそのまま実行しようとしても、動作が保証されない。ところが、OSの行なう処理をマクロ的に見ると、プログラムをロードしてからそれを実行するという流れがある。つまり、広

172

い範囲で見るとOSのプログラムにより命令コードを書き換えていることになる。また、特殊な用途では、プログラムによって生成した命令列を実行することもある。

この命令の目的は、そのような場合でも正しい命令の実行ができるようにすることである。すなわち、書き換えのあった命令コードに入る前にこの命令を実行しておけば、新しい命令コードが正しく実行されることが保証される。インプリメント上は、この命令によってパイプライン、命令キュー、命令キャッシュのページを行なうことになる。ただし、パイプラインやキャッシュのメカニズムがメモリの書き換えに対するバスモニタリング機構を持っており、メモリとの整合性がハードウェアで常に保証されていれば、必ずしもPIB命令でページを行なう必要はない。この場合、PIB命令はNOP命令として実行される。いずれにしても、この命令を実行した後に、パイプラインや命令キャッシュとメモリとの整合性が保証されていれば良いのである。MMUを用いて多重論理空間を実現している場合には、PIB命令を実行した論理空間に対してのみ書き換えた命令コードの実行が保証される。例えば、

context_A の命令コードを書き換え

STCTX

LDCTX context_B

context_B の命令コードを書き換え

PIB

といった命令列を実行した場合、context_Bでは変更された命令コードを実行しても動作が保証されるが、次にLDCTX context_Aを実行した後でも、context_Aの変更された命令コードの実行に対しては動作が保証されない。context_Aの命令実行を保証するためには、context_Aのコンテキストにおいて、もう一度PIB命令を実行する必要がある。これは、命令キャッシュにLSIDが導入された場合に、PIB命令では、LSIDの一致する命令キャッシュエントリをページするだけで済ませたいからである。PIB命令以外の命令では、いかなるジャンプ命令やOS関連命令(LDCTX, REIT, RRNG, TRAP, EIT起動など)を実行した後でも、命令コード書き換え部分のプログラムの動作は保証されない。これは、命令キャッシュのページをできるだけ減らすためである。したがって、OSがロードしたプログラムを最初に行なう時には、新しいコンテキストに入ってから（例えばLDCTX~REITの間で）、必ずPIBを実行する必要がある。この命令のニモニックPIB(Purge Instruction Buffer)の'Buffer'は、キャッシュやパイプラインなどを総括的に含めた意味で用いることばであり、PTLBの'B'のBufferに同じ用例がある。PIBというニモニックも、PTLBとの連想から作られたも

173

のである。この命令は特権命令ではない。ユーザプログラムからも使用できる。

〔命令コードの整合性について〕PIB命令の動作を正確に説明するため、「命令コードの整合性」という状態を以下のように定義する。「命令コードの整合性」とは、各論理空間の各論理アドレスについて、別々に定義される状態である。例えば、論理空間AではH' 00000000~H' 000fffffについて「命令コードの整合性」が保証され、論理空間BではH' 00010000~H' 0003ffffについて「命令コードの整合性」が保証されている、といった使い方をする。「命令コードの整合性」が保証されている領域の命令を実行した場合にのみ、正しい命令動作をする(executeのアクセス権チェックを含む)ことが保証される。一般には、「命令コードの整合性」の保証されている領域が命令コード領域であり、データ領域では「命令コードの整合性」が保証されていない。

・「命令コードの整合性」が保証されるようになるのは、次の場合である。

ーリセット時

物理空間(=論理空間)の全領域で「命令コードの整合性」が得られる。

ーPIB命令実行時

PIB命令を実行した論理空間の全領域で「命令コードの整合性」が得られる。AT=00の場合は、リセット時と同様、物理空間(=論理空間)の全領域で「命令コードの整合性」が得られる。

・「命令コードの整合性」が失われるようになるのは、次の場合である。

ーメモリ書き換え時

メモリ内容を書き換えた場合、書き換えた領域の「命令コードの整合性」は失われる。これは、論理アドレスによるメモリアクセスの場合も、物理アドレスによるメモリアクセスの場合(AT=00やLDP命令など)も同様である。

ーATE更新時

ATEを更新した場合、そのATEによりアドレス変換される領域の「命令コードの整合性」は失われる。したがって、例えば、LDATEでATE中の保護ビットを変更した場合にも、その後PIB命令を実行しなければ

保護情報のチェックが正しく行なわれないことになる。(これは、命令キャッシュと保護情報のチェックに関するインプリメントを軽くするために有効であろう)

以上の点に該当しない一般の命令実行ではBRA, JM

・予約命令例外

・RR='11'のとき

・BB='11'のとき

・EaRが@-SPのとき

・EaMfi, ShMfqiがRn, #imm_data, @SP+, @-SPのとき

174

P, JRNG, RRNG, TRAP, REIT, LDC TX, EIT起動などを含めて、「命令コード整合性の状態」は変化しない。

【0207】12-13. マルチプロセッサ用の命令
〔ニモニック〕

BSETI offset, base

〔命令の機能〕

【0208】

〔数13〕

10 bit → Z_flag, 1 → bit (interlocked)

【0209】ビットのセット (バスをロック)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図233に示す。

〔フラッグ変化〕図234に示す。

〔解説〕指定されたビットの値を反転したものをZ_flagにコピーし、その後そのビットを1にセットする。この2つの操作はバスをロックして行なわれ、不可分の操作になる。したがって、マルチプロセッサ間の同期をとるためにこの命令が使用できる。ShMfqi, EaMfiで指定されるアドレッシングモードでは、レジスタ直接モードRn, @-SP, @SP+, #imm_dataのモードは使用できない。アセンブラ表記では、メモリアクセスのサイズをbaseのサイズとして指定する。BSETI:Qでは、メモリアクセスのサイズは8ビットに固定されており、サイズは'B'のみを書くことができる。また、BSETI:G, BSETI:Eでのアクセスサイズ(baseのサイズ)。

30 H, .Wの指定は、BSET, BCLRと同じく<(L2)>とする。<(L2)>仕様でアクセスサイズ、

H, .Wを指定したのに、baseがアラインメントのとれていないアドレスであった場合には、メモリアクセスの範囲がインプリメント依存となる。これは、ビット操作命令と同様である。この時、インプリメントによって、アラインメントのとれていないワードやハーフワードのアクセスが行なわれる場合には、バスをロックしたまま複数のバスサイクルを実行する。これはCSI命令と同様である。本発明装置では<(L2)>となっているハーフワード、ワード単位のアクセスのインプリメントを行なう。またbaseとしてアラインメントのとれていないアドレスを指定した場合にも、アラインメントのとれたハーフワード、ワード単位でアクセスを行なう。

【0210】〔プログラム例外〕

175

〔ニモニック〕

BCLRI offset, base

〔命令の機能〕

【0211】

〔数14〕

-bit -> Z_flag, 0 -> bit (interlocked)

【0212】ビットのクリア (バスをロック)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図235に示す。

〔フラッグ変化〕図236に示す。

〔解説〕指定されたビットの値を反転したものをZ_flagにコピーし、その後そのビットを0にセットする。この2つの操作はバスをロックして行なわれ、不可分の操作になる。したがって、マルチプロセッサ間の同期をとるためにこの命令が使用できる。EaMfiで指定されるアドレッシングモードでは、レジスタ直接モードRn、@-SP、@SP+、#imm_dataのモードは使用できない。アセンブラ表記では、メモリアクセスのサイズをbaseのサイズとして指定する。BCLRI:G, BCLRI:Eでのアクセスサイズ(baseのサイズ)、H、.Wの指定は、BSET、BCLRと同じく〈〈L2〉〉とする。〈〈L2〉〉仕様でアクセスサイズ、H、.Wを指定したのに、baseがアラインメントのとれていないアドレスであった場合には、メモリアクセスの範囲がインプリメント依存となる。これは、ビット操作命令と同様である。この時、インプリメントによって、アラインメントのとれていないワードやハーフワードのアクセスが行なわれる場合には、バスをロックしたまま複数のバスサイクルを実行する。これはCSI命令と同様である。本発明装置では〈〈L2〉〉となっているハーフワード、ワード単位のアクセスのインプリメントを行なう。またbaseとし*

update ==> tmp

/*以下の動作はバスをロックして行なわれる*/

if (dest = comp)

then

tmp ==> dest

1 ==> Z_flag

else

dest ==> comp

0 ==> Z_flag

ビットパターン上の制約から、CSIでは、比較が成功しなくてもupdateの読みだしが行なわれる。また、CSI命令でのdestのアクセス権は、常にread, writeとも必要であるものとする。すなわちCSI命令で比較が失敗し、destに対して書き込みが起らない場合でも、destに対してwriteアク

176

*てアラインメントのとれていないアドレスを指定した場合にも、アラインメントのとれたハーフワード、ワード単位でアクセスを行なう。

【0213】〔プログラム例外〕

・予約命令例外

・RR='11' のとき

・BB='11' のとき

・EaR が@-SPのとき

・EaMfi がRn, #imm_data, @SP+, @-SPのとき

10

〔ニモニック〕

CSI comp, update, dest

〔命令の機能〕

compare and store (interlocked)

比較とストア (バスをロック)

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図237に示す。

20 〔フラッグ変化〕図238に示す。

〔解説〕destの値が以前の値(compにより指定)と同じであれば、内容を更新する命令である。この命令は、簡単な構造のデータをマルチプロセッサから更新していく場合に利用できる。CSI命令を実行した結果、destの値が以前の値と異なっていることがわかった場合、それは他のプロセッサがデータ内容を書き換えたことを意味している。したがって、CSI命令によってdestの値の食い違いを発見したプロセッサは、新しいdestの値をもとにして、そのデータ内容の更新をやり直さなければならない。このような方法をとることにより、マルチプロセッサの下でデータの一貫性を保つことができる。〔CSIのオペレーション〕

30

セス権がないとアドレス変換例外(ATRE)になる。RMC, EaMiRのサイズはRRで指定される。EaMiRで指定されるアドレッシングモードでは、@-SP, @SP+, Rn, #imm_dataのモードは使用できない。CSI命令で、サイズ、H、.Wを指定し、アラインメントの取れていないアドレスをオペラン

177

ドとした場合には、バスをロックしたまま複数のバスサイクルを実行する。この場合、read, writeのそれぞれが2回ずつのメモリアクセスに分かれるので、命令全体では、バスをロックしながらread, read, write, writeの4回のメモリアクセスを*

var1 EQU H'00000006 : アラインメントの取れていないアドレス

とした場合に、プロセッサAから

MOV.W #H'12345678, @var1

を実行し、プロセッサBから

MOV.W #H'87654321, @var1

を実行すると、メモリ書き込みのタイミングによっては、

H'00000006~7 =H'8765

H'00000008~9 =H'5678

となって、プロセッサAのMOV命令が先に実行された場合ともプロセッサBのMOV命令が先に実行された場合※

・予約命令例外

・RR='11' のとき

・BaR が@-SPのとき

・BaMiR がRn, #imm_data, @SP+, @-SPのとき

【0215】12-14. 制御空間、物理空間操作命令
本発明装置では、メインプロセッサの制御レジスタ群が、コプロセッサの制御レジスタ群やチップバス上の高速メモリなどとともに一つのアドレス空間を作ることができるようになっており、これを制御空間と呼ぶ。制御空間の考え方は、現在別チップとなっているコプロセッサやコンテキスト退避用の高速メモリが、将来メインプロセッサに内蔵された場合に、特に有効になる考え方である。制御レジスタ操作命令は、制御空間に対してアクセスを行なうための命令である。なお、LDC, STCなどの汎用的な制御空間操作命令は特権命令となっているため、ユーザが制御空間の一部であるPSB, PSMを操作するためには、LDPSB, STPSB, LDPSM, STPSM命令を使用する。本発明装置はアドレス変換機構を持たない。よって、論理空間アドレスと物理空間アドレスがつねに等しいため物理空間操作命令の機能は論理空間を操作する他の命令に吸収されてしまう。しかし、アドレス変換機構を持ち、論理空間と物理空間を区別して扱う本発明装置とのソフトウェア互換性を重視し、本発明装置では物理空間操作命令をサポートする。

〔ニモニック〕

LDC src, dest

〔命令の機能〕

load control space or register

制御空間へのロード

〔命令オプション〕なし

178

*行なうことになる。なお、CSI以外の一般命令で、アラインメントの取れていないアドレスに対してメモリアクセスを行なった場合には、バスはロックされない。したがって、例えば、

※合とも異なる結果になる可能性がある。マルチプロセッサ間の共有変数に対しては、通常データの書き込みだけではなくデータ更新(read-modify-write)を行なうのが普通なので、必然的にCSI命令を使うことになり、以上のような問題は発生しない。しかし、マルチプロセッサからCSI以外の命令でアラインメントのとれていない変数をアクセスする場合には、以上のような問題が生じることがあるので、注意しておく必要がある。

【0214】〔プログラム例外〕

〔命令ビットパターンとアセンブラ表記〕図239に示す。

〔フラッグ変化〕図240に示す。

〔解説〕srcの値を制御空間のdestに転送する。srcのサイズがdestより小さいときは、符号拡張される。dest/EaW%では、レジスタ直接モードRnの指定、@-SPの指定はできない。この命令は特権命令である。ring0以外から実行された場合には、特権命令違反例外(RIVE)となる。本発明装置では制御空間に対する、B, .Hのアクセス機能はサポートしない。制御空間としてはCPU内の制御レジスタのみをインプリメントする。また、UATB, SATBを実装していないためLDCによりUATB, SATBを変更することはできない。LDATE, STATE, LDP, STP, LDC, STC, MOVPA命令の中の特権空間を参照するオペランドにおいて、付加モードによりメモリの間接参照が起こった場合には、特殊空間の方ではなく論理空間(LS)の方を参照する。また、スタックポインタSPの参照があった場合には、PRNGではなく現在リングRNGのスタックが参照される。特殊空間のアドレスという意味を持つのは、最終的に得られた実効アドレスのみである。制御空間に対する、B, .Hのアクセス機能が全くないプロセッサにおいて、制御空間のオペランドのサイズとして、B, .Hを指定した場合には、予約命令例外(RIE)となる。未実装の制御レジスタ、または制御レジスタのないアドレスをLDCで指定した場合には、予約機能例外(RFE)となる。〈〈LV〉〉の領域についても同様であ

る。制御空間で利用できるアドレスに何らかの制限のあるプロセッサの場合、それに違反した場合には予約機能例外(RFE)とする。例えば、制御レジスタのアドレスを4の倍数に限るといった制限はこれに含まれる。コンテキスト退避用の高速メモリを内蔵したプロセッサであれば、制御レジスタ部分のアドレスのみが4の倍数に制限され、高速メモリ部分のアドレスは自由になるというケースが考えられるが、この場合にも、違反すると予約機能例外(RFE)になる。また、一部のアドレスについて、B、. Hの指定が可能なプロセッサにおいて、. B、. Hのアクセスができないアドレスを指定した場合にも、予約命令例外(RIE)ではなく予約機能例外(RFE)となる。これは、命令ビットパターン

(サイズ指定を含む)のみでエラーと判定できるものを予約命令例外(RIE)とし、アドレスやオペランド値によってエラーかどうかの状態が変化するものは予約機能例外(RFE)とする、という考え方に基づいたものである。制御空間のアドレスがチップ外(コプロセッサのアドレスなど)になり、インプリメントの制約によってその領域がアクセスできなくなっている場合にも、予約機能例外(RFE)が発生する。LDC, STCでは、制御空間のアドレスがコプロセッサのアドレスになった場合でも、コプロセッサ命令例外(CIE)は発生しない。コプロセッサ命令例外(CIE)が発生するのは、コプロセッサ用の命令を実行した場合に限られる。LDCで、制御レジスタの'-'、'+'で表現されるreservedのビットに異なる値を書き込もうとした場合や、あるフィールドに対してreservedの値を書き込もうとした場合には、予約機能例外(RFE)になる。PSWのSMRNGのフィールドに'001'などのreservedの値を書き込んだ場合も、これに含まれる。一方、'='、'#'で表現されているreservedのビットに異なる値を書き込んだ場合には、単に無視される。ただし、ユーザ向けのマニュアルでは、'='に対して必ず'0'を書き込んでもらうように注意しなければならない。また、'*'で表現されているビットには、何を書き込んでも単に無視される。このビットは、'='、'#'とは異なり、今後仕様を拡張した場合でも、使用されないことが保証されたビットである。したがって、LDCを実行する前に、このビットを'0'にマスクしておく必要はない。LDCでCTXBBを変更した場合には、メモリ上のCTXBBの内容と実際のチップ内のコンテキストとの整合性がとれなくなるが、これはプログラマの責任で処理する。ハードウェア的には、単にCTXBBの変更のみを行なう。CTXBBの変更とコンテキストのロードを両方行なう場合は、LDCTXを使用すればよい。LDC命令によってUATB, SATBが変更される時は、それに伴ってTLBや論理キャッシュのページ(PSTLB/A Tに相当する処理)が自動的に行なわれる。LSID

を実装したプロセッサの場合は、LSID制御レジスタにより指定される論理空間がページの対象となる。この場合、LDC命令には、PSTLB命令のような/SS, /ASのオプションは設けられていないが、これは次のような理由によっている。PTLB, PSTLB命令によるTLBのバージの場合は、LDC *, UATBの場合とは異なり、他の論理空間のキャッシュやTLBもバージできるように、LSIDの機能に相当するパラメータを、別のレジスタ(R1)によって指定している。この場合、LSIDの制御レジスタは使用しない。したがって、そのパラメータを使用するかどうかを区別するために、/SS, /ASのオプションを切り換える必要がある。ところが、LDC *, UATBの場合は、データの矛盾をなくするために、現在使用中の空間に対してキャッシュやTLBのバージを行なうのであるから、LSIDの制御レジスタは本来の意味で働く。つまり、一般のメモリアクセスと同様に、LSID制御レジスタによって指定された論理空間がページの対象となる。LSID未実装のプロセッサでは、全論理空間(一つだけであるが)がページの対象となる。

【0216】[プログラム例外]

・予約命令例外

- ・RR='11' のとき
- ・WW='10' 以外のとき
- ・EaR が0-SPのとき
- ・EaW%がRn, #imm_data, 0SP+, 0-SPのとき

・特権命令違反例外

- ・ring 0 以外から実行されたとき
- ・予約機能例外
- ・未実装の制御レジスタをアクセスしたとき
- ・制御レジスタの特定フィールドに対してreservedの値を書き込もうとしたとき(=, #, *は除く)
- ・EaW%のアドレスのワードアラインメントがとれていないとき

[ニモニック]

STC src, dest

[命令の機能]

store control space or register

制御空間からのストア

[命令オプション] なし

[命令ビットパターンとアセンブラ表記] 図241に示す。

[フラグ変化] 図242に示す。

[解説] 制御空間にあるsrcの値をdestに転送する。STCでは、srcとdestのサイズが共通に指定されるため、異種サイズ間の転送は行なわれない。この命令は特権命令である。ring 0 以外から実行された場合には、特権命令違反例外(PIVE)となる。s

181

rc/EaR%では、レジスタ直接モードRnの指定、イミディエート#imm_dataの指定、@SP+の指定はできない。本発明装置では、制御空間に対する、B、. Hのアクセス機能はサポートしない。制御空間についてはCPU内の制御レジスタのみをインプリメントする。LDATE, STATE, LDP, STP, LDC, STC, MOVPA命令の中の特殊空間を参照するオペランドにおいて、付加モードによりメモリの間接参照が起こった場合には、特殊空間の方ではなく論理空間(LS)の方を参照する。また、スタックポインタSPの参照があった場合には、PRNGではなく現在リングRNGのスタックが参照される。特殊空間のアドレスという意味を持つのは、最終的に得られた実効アドレスのみである。制御空間に対する、B、. Hのアクセス機能が全くないプロセッサにおいて、制御空間のオペランドのサイズとして、B、. Hを指定した場合には、予約命令例外(RIE)となる。未実装の制御レジスタ、または制御レジスタのないアドレスをSTCで指定した場合には、予約機能例外(RFE)となる。〈〈LV〉〉の領域についても同様である。制御空間で利用できるアドレスに何らかの制限のあるプロセッサの場合、それに違反した場合には予約機能例外(RFE)とする。例えば、制御レジスタのアドレスを4の倍数に限るといった制限はこれに含まれる。コンテキスト退避用の高速メモリを内蔵したプロセッサであれば、制御レジスタ部分のアドレスのみが4の倍数に制限され、高速メモリ部分のアドレスは自由になるというケースが考えられるが、この場合にも、違反すると予約機能例外(RFE)になる。また、一部のアドレスについてのみ、B、. Hの指定が可能なプロセッサにおいて、. B、. Hのアクセスができないアドレスを指定した場合にも、予約命令例外(RIE)ではなく予約機能例外(RFE)となる。これは、命令ビットパターン(サイズ指定を含む)のみでエラーと判定できるものを予約命令例外(RIE)とし、アドレスやオペランド値によってエラーかどうかの状態が変化するものは予約機能例外(RFE)とする、という考え方に基づいたものである。制御空間のアドレスがチップ外(コプロセッサのアドレスなど)になり、インプリメントの制約によってその領域がアクセスできなくなっている場合にも、予約機能例外(RFE)が発生する。LDC, STCでは、制御空間のアドレスがコプロセッサのアドレスになった場合でも、コプロセッサ命令例外(CIE)は発生しない。コプロセッサ命令例外(CIE)が発生するのは、コプロセッサ用の命令を実行した場合に限られる。STCで、制御レジスタの'-'で表現されているビットを読みだした場合には'0'が、'+'のビットを読みだした場合には'1'が読み出される。また、'=', '#', '*'のビットを読み出そうとした場合に得られる値は、不定である。インプリメントによって、'0'固定の場合、'1'固

182

定の場合、以前に書き込んだ値がそのまま読み出される場合がある。将来の拡張のため、'=', '#', '*'のビットの値を利用したプログラミングは行なわないように、ユーザ向けのマニュアルに明記しなければならない。

【0217】〔プログラム例外〕

・予約命令例外

・WW='10'以外のとき

・EaR%がRn, #imm_data, @SP+, @-SPのとき

・EaWが#imm_data, @SP+のとき

・特権命令違反例外

・ring 0以外から実行されたとき

・予約機能例外

・未実装の制御レジスタをアクセスしたとき

・EaR%のアドレスのワードアラインメントがとれていないとき

〔ニモニック〕

20 LDP SB src

〔命令の機能〕

load PSB

PSBへのロード

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図243に示す。

〔フラッグ変化〕図244に示す。

〔解説〕srcをPSBに転送する。ユーザのコールテンなどで、PSB, PSMの個々のビットの意味とは関係なく退避や復帰を行なう場合を除けば、PSM, PSBでは、一部のフィールドのみの書き換えをしたいということが多い。そのため、LDP SB, LDP SM命令のsrcオペランドは16ビット(EaRh)となっており、上位バイトがマスク(変更されるビットを0とする)、下位バイトが変更データを表わすという仕様になっている。つまり、srcを

src = {S0. S1... S7. S8. S9... S15}

とすると、

〔LDP SBのオペレーション〕

40 【0218】

【数15】

{S0. S1... S7}. and. PSB). or. (-{S0. S1... S7}

. and. {S8. S9... S15}) ==>PSB

ただし'-'はビット否定

【0219】となる。例えば、2⁴の位置にあるX__flagをセットする命令は、

LSPSB #H'ef10

50 となる。上位バイトで、変更されるビットの方を0、変

更されない方を1としたのは、変更される方をデフォルトと考える方が自然だと判断したからである。8ビット全部を変更する場合には、上位バイトをすべて0にして単なるバイトデータを書けばよい。8ビット全部の変更は、最初に述べたように、ユーザ側でPSB、PSMの退避や復帰をする場合に必要である。LDPSB、LDPSMで、PSB、PSMの未使用フィールドの値を"1"にしようとした場合には、予約機能例外(RFE)が発生する。

【0220】〔プログラム例外〕

- ・予約命令例外
 - ・EaRhが@-SPのとき
- 〔ニモニック〕

LDPSM src

〔命令の機能〕

load PSM

PSMへのロード

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図245に示す。

〔フラッグ変化〕図246に示す。

〔解説〕srcをPSMに転送する。ユーザのコルーチンなどで、PSB、PSMの個々のビットの意味とは関係なく退避や復帰を行なう場合を除けば、PSM、PSBでは、一部のフィールドのみの書き換えをしたいということが多い。そのため、LDPSB、LDPSM命令のsrcオペランドは16ビット(EaRh)となっており、上位バイトがマスク(変更されるビットを0とする)、下位バイトが変更データを表わすという仕様になっている。つまり、srcを

src = {S0, S1... S7, S8, S9... S15}

とすると、

〔LDPSMのオペレーション〕

【0221】

〔数16〕

{S0, S1... S7}.and. PSM.or. (~{S0, S1... S7}.and. {S8, S9... S15})=>PSM

ただし'-'はビット否定

【0222】となる。LDPSB、LDPSMで、PSB、PSMの未使用フィールドの値を"1"にしようとした場合には、予約機能例外(RFE)が発生する。

【0223】〔プログラム例外〕

- ・予約命令例外

・EaRhが@-SPのとき

〔ニモニック〕

STPSB dest

〔命令の機能〕

store PSB

PSBからのストア

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図247に示す。

〔フラッグ変化〕図248に示す。

〔解説〕PSBをdestに転送する。上位8ビットは必ず0となる。destが8ビットではなく16ビットとなっており、上位8ビットが常に0を返すようになっているのは、LDPSM、LDPSBでそのままPSM、PSBの復帰ができるように配慮したためである。

【0224】〔プログラム例外〕

- ・予約命令例外

・EaWhが#imm_data, @SP+ のとき

〔ニモニック〕

STPSM dest

〔命令の機能〕

store PSM

PSMからのストア

20 〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図249に示す。

〔フラッグ変化〕図250に示す。

〔解説〕PSMをdestに転送する。上位8ビットは必ず0となる。destが8ビットではなく16ビットとなっており、上位8ビットが常に0を返すようになっているのは、LDPSM、LDPSBでそのままPSM、PSBの復帰ができるように配慮したためである。

【0225】〔プログラム例外〕

- ・予約命令例外

30

・EaWhが#imm_data, @SP+ のとき

〔ニモニック〕

LDP src, dest

〔命令の機能〕

load physical space

物理空間へのロード

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図251に示す。

〔フラッグ変化〕図252に示す。

〔解説〕srcの値を物理空間のdestに転送する。srcのサイズがdestより小さいときは、符号拡張される。本発明装置はアドレス変換機構を持たないので論理空間アドレスと物理空間アドレスがつねに等しく、この命令の機能はMOV命令に吸収されてしまう。しかし、アドレス変換機構をもち論理空間と物理空間を区別して扱う本発明装置とのソフトウェア互換性を取るためこの命令をサポートする。この命令は特権命令である。

50 dest/EaW%では、レジスタ直接モードRnの指

185

定、@-SPの指定はできない。LDATE, STATE, LDP, STP, LDC, STC, MOVPA命令の中の特殊空間を参照するオペランドにおいて、付加モードによりメモリの間接参照が起こった場合には、特殊空間の方ではなく論理空間(LS)の方を参照する。また、スタックポインタSPの参照があった場合には、PRNGではなく現在リングRNGのスタックが参照される。特殊空間のアドレスという意味を持つのは、最終的に得られた実効アドレスのみである。

【0226】〔プログラム例外〕

・予約命令例外

- ・RR='11' のとき
- ・WW='11' のとき
- ・EaR が@-SPのとき
- ・EaW%がRn, #imm_data, @SP+, @-SPのとき

・特権命令違反例外

・ring 0以外から実行されたとき

〔ニモニク〕

STP src, dest

〔命令の機能〕

store physical space

物理空間からのストア

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図253に示す。

〔フラッグ変化〕図254に示す。

〔解説〕物理空間にあるsrcの値をdestに転送する。STPでは、srcとdestのサイズが共通に指定されるため、異種サイズ間の転送は行なわれない。本発明装置はアドレス変換機構を持たないので論理空間アドレスと物理空間アドレスがつねに等しく、この命令の機能はMOV命令に吸収されてしまう。しかし、アドレス変換機構を持ち論理空間と物理空間を区別して扱う本発明装置とのソフトウェア互換性を取るためこの命令をサポートする。この命令は特権命令である。src/EaR%では、レジスタ直接モードRnの指定、イミディエート #imm_dataの指定、@SP+の指定はできない。LDATE, STATE, LDP, STP, LDC, STC, MOVPA命令の中の特殊空間を参照するオペランドにおいて、付加モードによりメモリの間接参照が起こった場合には、特殊空間の方ではなく論理空間(LS)の方を参照する。また、スタックポインタSPの参照があった場合には、PRNGではなく現在リングRNGのスタックが参照される。特殊空間のアドレスという意味を持つのは、最終的に得られた実効アドレスのみである。

【0227】〔プログラム例外〕

186

・予約命令例外

- ・WW='11' のとき
- ・EaR%がRn, #imm_data, @SP+, @-SPのとき
- ・EaW が#imm_data, @SP+ のとき

・特権命令違反例外

・ring 0以外から実行されたとき

【0228】12-15. OS関連命令

10 〔ニモニク〕

JRNG vector (「本発明装置」ではサポートしない)

〔命令の機能〕

jump to new ring

リング間コール

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図255に示す。

〔フラッグ変化〕図256に示す。

20 〔解説〕リング間の遷移とジャンプ(リング間コール)を行なう。この命令は、現在のリングよりも内側のリングレベルにあるプログラムの呼び出し(システムコールの呼び出しを含む)を行なうために使用される。外側のリングから内側のリングを保護するため、JRNGでのジャンプ先は特定のアドレスに制限されている。このアドレスを入れたテーブルをリング間遷移テーブルJRNGVT(JRNG Vector table)と呼ぶ。JRNG命令では、vectorオペランドがJRNGVTに対するインデクスとなる。JRNGVTの一つのエントリを、JRNGVTEと呼ぶ。JRNGVTは、vectorに対するエントリを65535個持つテーブルであり、そのベースの論理アドレスがJRNGVBによって示される。vectorのサイズは16ビットとなっている。JRNGVBは制御レジスタの一つであり、次のような構成になっている。

【0229】・JRNGVB

JRNGVBは、図257に示すようにJRNG命令のベクトルテーブルの先頭の論理アドレスを示す。テーブルのベースアドレスは、アラインメントのため、下位の3ビットが0に固定される。Eが0の時はJRNGは実行禁止であり、JRNGを実行するとリング遷移違反例外(RTVE)となる。この時、JRNGVBは意味を持たないので、OSはこのフィールドを自由に使用して良い。'='のビットには'0'を入れておかなければならない。ただし、このビットが0でなくても単に無視される。JRNGVTEは8バイトであり、図258のような構成になっている。これは、内側のリングへ入るためのゲートという意味合いをもったものである。

50 ・ARの機能は、そのvectorで示されるエントリのリング間コールが、最低でどのリングから発行可能で

あるかを示すものである。ARで示されるリングよりも現在のリングの方が外側の場合には、リング間コール（システムコール）が許可されていないものと見なされ、リング遷移違反例外（RTVE）となる。ARは、その意味から考えて、PSWのPRNGの位置に相当するフィールドを使う。これは、JRNGVT、EITVTの各エントリが、基本的にはPSW+PCのサブセットの形になっているという考え方に基づいたものである。

・VXの機能は、OSとユーザプログラムとの間で32/64ビットのモードが異なっている場合に有効である。

・JRNGVTEの未使用フィールド（'='で示される）、および'VX'ビットには、'0'を入れておかなければならない。ただし、実際には、これらのビットが'1'であったとしても、単に無視されるだけである。これを予約機能例外（RFE）としないのは、インプリメントの負担を考えたためである。

〔詳細仕様調整中〕

・また、JRNGVTEのVPCのフィールドは偶数でなければならない。つまり、VPCのフィールドのLSBは'0'でなければならない。違反した場合には、JRNG実行時に奇数アドレスジャンプ例外（OAJE）が発生する。〔詳細仕様調整中〕

【0230】JRNGVBはMSB=0の時UATB、MSB=1の時SATBを使ってアドレス変換される。JRNGVBのアドレスを論理アドレスとしたのは、次のような利点があるためである。

①テーブルをコンテキスト毎に持つことが可能

②テーブルの仮想化が可能である。すなわち、テーブル自体をページ不在にすることができる。

③EITであるTRAPAとの性格の違いが、よりはっきりする。

JRNGVBを論理アドレスと考えることにより、テーブルの仮想化が可能となる。「本発明装置」では、ベクトルが16ビット（65536エントリ、512KBテーブルサイズ）で非常に多くなっており、しかもベクトルの上限を指定するレジスタが設けられていない。しかし、JRNGVBが論理アドレスとなっているため、MMU機能との組み合わせが可能であり、必ずしもテーブル分の物理メモリを負担する必要はなくなっている。JRNGVT部分のSTE、PTEを未使用領域にしておけば、16ビット=65536エントリ分のテーブルをすべて物理メモリで用意する必要はない。JRNGVTEの読みだしは、論理アドレスを用いた一般のメモリアクセスと同様に行なわれる。したがって、JRNGVTEは、JRNGを実行したプログラムのリングのアクセス権で読み出される。JRNGを実行したリングから、指定したベクトルのJRNGVTEをreadする権利が

ない時は、アドレス変換例外（ATRE）のリング保護違反エラーになる。また、指定したベクトルのJRNGVTEが未使用領域となっていた場合には、アドレス変換例外（ATRE）の未使用領域参照エラーとなる。使用する側から見ると、これらのエラーはリング遷移違反例外（RTVE）と同じように扱われる方が望ましいのであるが、主としてインプリメントの都合により、上記のような仕様になっている。JRNGVTEの読み出しの際には、このほかページ不在例外（POE）、バスアクセス例外（BAE）などの発生する可能性がある。JRNG機能の導入により、JRNGVTのための論理空間を必ず512KB消費することになる。また、不当なリング間コールを防ぐためには、ユーザプログラムを実行する前に、OSがJRNGVTの領域のSTE、PTEのセットを行なっておく必要がある。そこで、リング間コール機能を使用しない場合に、こういった処理が全く不用になるように、リング間コール機能全体をディスエーブルにすることができるようになっている。この指定には、JRNGVBのLSBのEビットを使う。JRNGVBのEビットが0の場合は、リング間コール機能は使用できなくなり、JRNGを実行した場合には無条件にリング遷移違反例外（RTVE）となる。

【0231】結局、JRNGが成功するには次の条件を満たさなければならない。

——JRNGVBのE=1。

E=0であれば、JRNGVTが用意されていないことを意味するのでリング遷移違反例外（RTVE）となる。

——指定したベクトルに対するJRNGVTEが、JRNG実行前のリングからread可能。

ページ不在例外が発生した場合は、ページインの後再実行することになる。アドレス変換例外（ATRE）の未使用領域参照エラーが発生した場合は、そこまでテーブルが用意されていないことを意味するので、そのユーザプログラムにエラーを返すのが普通である。readのアクセス権がない場合は、保護の観点からJRNGの実行を禁止していることを意味するので、やはりそのユーザプログラムにエラーを返すのが普通である。これはVAフィールドと同等の意味になるが、512ベクトル毎の指定となる。

——現在リング \geq VRが成立。

外側のリングには行けない。違反した場合はリング遷移違反例外（RTVE）となる。

——現在リング \leq ARが成立。

受け付け可能リングのチェック。違反した場合はリング遷移違反例外（RTVE）となる。なお、ARはJRNGVTEのARフィールドである。

【0232】〔JRNGのオペレーション〕

189

190

if JRNGVB の E ビット=0 then リング遷移違反例外(RTVE)

論理アドレス $mem[vector \times 8 + JRNGVB]$ より VR, AR, VPC をフェッチ

if $旧RNG > AR$. or. $旧RNG < VR$ then リング遷移違反例外(RTVE)

旧SP ==> TOS↓ (VRで示される新しいスタックを使用)

旧PC ==> TOS↓

旧PCとしては、JRNG命令の次の命令の先頭アドレスがスタックにプッシュされる。

これは、JSR 命令と同じである。

旧PSW . and. $B'01110000_00000000_11111111_11111111$ ==> TOS↓

旧PSW は、RRNGで意味を持つフィールド、つまり、RNG, XA, PSW のフィールドのみがそのままスタックにプッシュされ、それ以外のSM, AT, IMASK のフィールドは0にマスクされてからスタックにプッシュされる。これは、外側のリングのプログラムが、OSのみが知っているべき情報(IMASKなど)を読み出せないようにするためである。

旧RNG ==> 新PRNG

VR ==> 新RNG

VPC ==> 新PC

JRNG命令により形成されるスタックフレームは、図259のようになる。

【0233】ここで、旧リングのSPを新リングのスタックに積んだのは、新リングから旧リングのスタックポインタSPやスタックをアクセスするためである。リング毎のスタックは制御レジスタとしてアクセス可能であるが、アクセスするためには特権命令(STC)を利用しなければならない。したがって、例えばring1からring3のスタックに積まれたパラメータを見たいという場合には、この機能が必要である。JRNGでは、PSSの一部とPSMのPRNGのみが更新され、PSBは更新されない。また、リング間コールの機能では、EITとは異なり、スタックフォーマットは一種類しかないで、FORMAT(EITINF)はスタックに積まれない。JRNG:Eの場合は、vectorがゼロ拡張される。AT=00 (アドレス変換なし)の場合は、JRNGVBは物理アドレスを表わす。JRNG実行時に、リング遷移違反例外(RTVE)などの命令再実行型のEITが発生した場合、JRNG固有の機能であるリング間コール用のスタックフレームの形成は行なわれず、EIT処理用のスタックフレームのみが形成される。例えば、SMRNG=000の状態ではJRNGを実行し、RNG=00にジャンプしようとしたが、何らかのエラーによりEITが発生した場合は、図260のようなスタックフレームになる。図261ではない。

(A)のような仕様になっているのは、EIT発生後の命令再実行のことを考えているからである。つまり、EIT処理ハンドラに入る前に、プロセッサの状態をでき

るだけ命令実行前の状態に戻そうとしているからである。

EITで使用するスタックとJRNGで使用するスタックが異なっていた場合は、EIT発生時にEITで使用するスタックのみが変化し、JRNGで使用するスタックのSPは変化しない。

【0234】JRNGでは、現在リングと同じリングにジャンプすることも可能である。この場合、JRNGでスタックの切り換えは起こらない。SPとしてスタックにプッシュされる値は、命令実行前のSPの値になる。これは、JRNG命令の最初でPUSHSPを実行したのと同じ動作である。この様子を図262に示す。JRNGで現在リングと同じリングにジャンプする場合に、JRNG:Gのvectorオペランドがメモリ上にあり、それがJRNG命令の実行に伴って形成されるスタックフレーム領域と重なっていると、命令再実行がきわめて難しくなる。そこで、JRNG:G命令では、メモリアクセスを伴うアドレッシングモード、つまりレジスタ直接Rnとイミディエート以外のアドレッシングモードは、すべて禁止している。この命令のオペランドとして動的な値を設定したい場合には、テンポラリレジスタを一つ用意し、レジスタ直接Rnのモードを利用する必要がある。リング間コールの機能はEITには含まれない。TRAPAとJRNGは、どちらもOSのシステムコールの呼出しを行なうことを目的にした命令である。一般には、BTRONのようにシステムコールの数が多く、複数のリングを利用するOSではJRNGを利用し、ITRONのようにシステムコールの数があまり多くなく、2リングで十分なOSではTRAPAを利用す

191

ることになる。TRAPAでは、ring1, ring2に行くことはできないので、BTRONで外核をring1に置く場合には、当然J RNGを使う必要がある。

【0235】なお、OSをユーザが拡張するような場合には、外向きのリング間コールが必要になることがあり、BTRONでもそういった例がある。しかし、頻度*

・予約命令例外

・P='1' のとき

・EaRh!MがRn, fimm_data以外のとき

・〈〈L1〉〉機能例外

・J RNGの正しいビットパターンがデコードされるとき

〔ニモニック〕

RRNG (「本発明装置」ではサポートしない)

〔命令の機能〕

return from previous ring※

↑TOS ==> temp1

↑TOS ==> temp2

↑TOS ==> SP of temp1<RNG>

if RNG> temp1<RNG> thenリング遷移違反例外(RTVE)起動

・temp1<RNG>は、temp1 を PSWと見た時にRNG フィールドに相当する部分を表わす。

・このチェックがないと、RRNG命令で不正に内側のリングに入ることが可能になる。

temp1<PSH> ==> PSH (PRNGを含む)

temp1<RNG> ==> RNG

temp1<XA> ==> XA

temp2 ==> PC

このほか、RRNG命令実行時にはDCEのEITが起動される場合があり、そのチェックを行なう必要がある。詳しくは付録9を参照。ここで、PRNGの旧のスタックポインタをRNGのスタックからポップして、再びPRNGのスタックポインタとして設定しているのは、PRNGのスタックに積まれたシステムコールのパラメータのポップなどにより、OS側からユーザのスタックポインタを更新することを想定したためである。RRNGで内側のリングに入ろうとした場合には、リング遷移違反例外(RTVE)となる。また、スタックからポップされたPCが奇数であった場合には、奇数アドレスジャンプ例外(OAJE)となる。現在のPSWのSMが0で、RRNG命令によりポップされるスタック中のRNG(上記オペレーション中のtemp1<RNG>)が0でない場合は、RRNG命令の実行によって、PSW中のSMとRNGの組み合わせがreservedのパターンとなる。この場合には、予約機能例外(RF

192

*としては多くないこと、リング保護の趣旨とは矛盾していること、完全な保護を行なうためにはリターンスタックを別に設ける必要があり、インプリメントの負担が大きくなること、などの理由により、命令セットのレベルでは外向きのリング間コールをサポートしない。

【0236】〔プログラム例外〕

※リング間リターン

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図263に示す

〔フラグ変化〕図264に示す

〔解説〕リング間コールに対するリターンを行なう。

【0237】〔RRNGのオペレーション〕

E)となる。

【0238】RRNG命令で、命令再実行型の例外であるRTVE, RFEが発生した場合には、RRNG命令実行後に無くなる予定であったリング間コールのスタックフレームが、そのまま残る。したがって、EITとリング間コールで同じスタックを使っていた場合には、そのスタックフレームに追加される形でEITのスタックフレームが形成される。また、EITとリング間コールで異なるスタックを使っていた場合には、リング間コールで使用していたスタックの内容やスタックポインタは変化しない。この点は、RRNGでDCEを起動する場合とは異なっている。DCEの場合には、前のリング間コールのスタックフレームをクリアしてから、DCEの新しいスタックフレームを構成する。

〈〈RFE発生時のスタックの例-EITで同じスタックを利用する場合〉〉図265に示すこれに対して、OAJEは命令完了型のEITとなる予定である。〔詳細

193

仕様調整中]

その場合は、DCEと同じように、リング間コールのスタックフレームをクリアしてからEITのスタックフレームを生成するということになる。RRNG命令でOAJEが発生する場合のスタックの動きは、次のようになる。

〈〈OAJE発生時のスタックの例—EITで同じスタックを利用する場合〉〉

(RRNG実行前) 図266に示す

(RRNG実行後、OAJE起動後) 図267に示す
RRNG命令でスタック中からポップされたPSW(上記のtemp1)のうち、PSH, RNG, XA以外のフィールドは、無視される。ただし、プログラミング上は、JRNG命令からRRNG命令までの間で、スタック中に退避されたPSWのPSH, RNG, XA以外のフィールドを書き換えてはいけない。RRNG命令(32ビット)で同じリングに戻る場合、SPの最終値は
mem[initSP + 8] ==> SP

(ただし、+8はPC.PSWの分)

となる。これは、PC, PSWの処理の後POP SPを実行したのと同じ動作である。JRNGVBのEビットは、RRNG命令の動作には関係しない。Eビットが0の場合にも、RRNG命令の実行は行なわれる。

【0239】〔プログラム例外〕

・予約命令例外

・P='1' のとき

・〈〈L1〉〉機能例外

・RRNGの正しいビットパターンがデコードされたとき

〔ニモニック〕

TRAPA vector

〔命令の機能〕

TRAP always

ソフトウェア割り込み、トラップ

・予約命令例外

・P='1' のとき

・cccc='1110, 1111' のとき

・条件トラップ命令

〔ニモニック〕

REIT

〔命令の機能〕

return from EIT

EIT処理(割り込み、例外処理)からのリターン

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図272に示す

194

*〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図268に示す

〔フラッグ変化〕図269に示す

〔解説〕内部割り込み(トラップ)が発生する。この命令は、ユーザプロセスからOSを呼ぶ場合などに利用する。TRAPA命令ではEITが起動されるので、必ずリング0に入ることになる。TRAP, TRAPAでは、その他のEIT処理と同じく、PSSの一部およびPSMのPRNGのみが更新される。PSMのPRNG以外のフィールド(PSBを含む)は更新されない。

【0240】〔プログラム例外〕

・予約命令例外

・P='1' のとき

・無条件トラップ命令

〔ニモニック〕

TRAP

〔命令の機能〕

TRAP conditionally

20 条件によるソフトウェア割り込み、トラップ

〔命令オプション〕

/条件指定各種(cccc)

〔命令ビットパターンとアセンブラ表記〕図270に示す

〔フラッグ変化〕図271に示す

〔解説〕指定された条件が満たされていた場合に内部割り込み(トラップ)が発生する。TRAP命令ではEITが起動されるので、必ずリング0に入ることになる。条件の指定方法は、Bcc命令と同じである。TRAP, TRAPAでは、その他のEIT処理と同じく、PSSの一部およびPSMのPRNGのみが更新される。PSMのPRNG以外のフィールド(PSBを含む)は更新されない。TRAPで未定義の条件を指定した場合には、予約命令例外(RIE)となる。

* 【0241】〔プログラム例外〕

40 〔フラッグ変化〕図273に示す

〔解説〕「本発明装置」では、例外、外部割り込み、内部割り込みを総称してEIT(Exception, Interrupt, Trap)と呼ぶ。REIT命令は、EITからのリターン、すなわち、OSからのリターンや割り込み処理からのリターンを行なうために使用する命令である。この命令は特権命令である。

【0242】〔REITのオペレーション〕

195

↑TOS ==> PSW;
 ↑TOS ==> FORMAT/VECTOR;
 ↑TOS ==> PC;

このほか、EIT のタイプによっては、スタックに付加情報が積まれている場合がある。それをポップしてEIT 発生前の状態に戻す。追加情報の有無は、FORMAT/VECTOR(EITINF) で判定する。また、REIT 命令実行時には、DI、DCEのEIT が起動される場合があり、そのチェックを行なう必要がある。詳しくは付録9を参照。

FORMAT/VECTORとして、サポートされないスタックフォーマットが指定されていた場合には、予約スタックフォーマット例外(RSFE)となる。この場合、フォーマットが不当であったスタックフレームは、追加情報の有無が判定できないためにそのまま残し、そのスタックフレームに追加される形でRSFEのスタックフレームが形成される。この点は、REITでDI、DCEを起動する場合とは異なっている。DI、DCEの場合には、前のEITのスタックフレームをクリアしてから、DI、DCEの新しいスタックフレームを構成

する。
 (〈RSFEの処理—RSFEで同じスタックを利用する場合〉)図274に示す

REIT命令で、スタックからポップされたPCが奇数であった場合には、奇数アドレスジャンプ例外(OAJE)となる。また、スタックからポップされたPSWによって、PSW内のreserved('ー')のビット(XAビットを含む)を'1'に書き換えようとした場合や、SMRNGとしてreservedの値を書き込もうとした場合には、予約機能例外(RFE)となる。SMビットの変化については特にチェックしない。EITから戻るためにREIT命令を使う限り、REIT命令の実行によってSMが1から0に変わることはないはずである。しかし、これは運用上の問題として対応することにし、REIT命令ではSMが1から0になったかどうかのチェックは行なわない。

【0243】〔プログラム例外〕

- ・予約命令例外
- ・P='1' のとき
- ・特権命令違反例外

196

- ・ring 0以外から実行されたとき
- ・予約スタックフォーマット例外
- ・EITから復帰する際に、サポートされていないスタックフォーマットが指定されたとき
- ・奇数アドレスジャンプ例外
- ・スタックからポップされたPCが奇数であったとき
- ・予約機能例外
- ・スタックからポップされたPSWによって、PSW内にreservedの値を書き込もうとしたとき

【0244】〔ニモニック〕

WAIT imask

〔命令の機能〕

set IMASK and wait

停止、割り込み待ち

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図275に示す

〔フラッグ変化〕図276に示す

〔解説〕PSWのIMASKフィールドをセットし、プログラムの実行を停止する。外部割り込みまたはリセットにより実行を再開する。この命令は特権命令である。imaskは、符号なしの数として解釈される。imask ≥ 16の場合には、予約機能例外(RFE)を発生する。外部割り込みの場合には、割り込みが起るまで確定できない情報(SP1/SP0のスタックの選択、ベクトル番号)がある。したがって、WAIT命令では、外部割り込みが発生してから情報をスタックに退避する。

【0245】

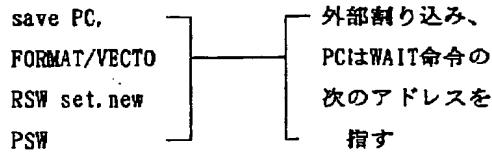
〔表1〕

[WAITのオペレーション]

imask ==> IMASK

wait for interrupt

<== 外部割り込み



【0246】 [プログラム例外]

- ・予約命令例外
- ・—' 1' のとき
- ・特権命令違反例外
- ・ring 0 から実行されたとき
[ニモニク]

LDCTX ctxaddr

[命令の機能]

load context from CTXB
コンテキストのロード

[命令オプション]

/LS 論理空間からCTXBをロード

/CS 制御空間からCTXBをロード (〈L
2〉)

[命令ビットパターンとアセンブラ表記] 図277に示す

[フラッグ変化] 図278に示す

[解説] ctxaddr で示される実効アドレスを CTXB レジスタにロードし、タスクやプロセスのコンテキストブロック (CTXB) の内容をプロセッサのレジスタにロードする。ロードの行なわれるレジスタは、MMUの有無やCTXB F Mの内容によって変化するが、SP0~SP3, UATB, CSWなどが含まれる。LDCTXで転送されるレジスタの詳細については、付録8を参照のこと。

【0247】 /LS オプションを指定した場合には、ctxaddr は論理空間のアドレスを意味する。この場合には、論理空間上にCTXB が置かれていることになる。また、/CS オプションを指定した場合には、ctxaddr は制御空間のアドレスを意味する。このオプションは、将来コンテキスト退避用の高速メモリをチップに内蔵した場合に使用することを想定したものであり、現在は〈〈L2〉〉となっている。これらのオプションは、チップやチップバスのインプリメントに合わせて最も高速なコンテキストスイッチを行なうため、CTXBを置く空間に自由度を持たせるという目的で設けられたものである。「本発明装置」では/CS オプションはサポートしない。「本発明装置」標準のMMUを内蔵したプロセッサでは、LDCTX命令でUATBの変更

が行なわれる。この場合、LSIDを実装しないプロセッサであれば、UATBの変更に伴ってTLBやキャッシュのバージ (PSTLB/ATに相当する処理) が自動的に行なわれる。LDCTX命令では論理空間の切り換えが行なわれるため、LDCTX/LSが意味のある動作を行なうには、ctxaddrはSRを指している必要がある。LDCTX/LSでctxaddrがURを指していた場合の動作は保証されない。「本発明装置」のLDCTX, STCTX命令では、汎用レジスタR0~R14の転送を行なっていない。これは、次のような理由による。

【0248】 汎用レジスタについては、LDM, STM命令で転送することが可能であり、LDM, STMならばレジスタの指定も可能である。実際のコンテキストスイッチの処理では、入れ換えを行なうレジスタ以外にワーキング用のレジスタが必要になることが多いので、一部のレジスタは転送しない方がよい場合がある。したがって、LDM, STMのような、より汎用的な命令を使用する方が適当である。

—現在は、まだコンテキスト退避用のメモリをチップに内蔵することが技術的に難しく、コンテキストの退避には外部メモリを利用せざるを得ない。その場合、LDCTXで汎用レジスタの転送まで行なっても、汎用レジスタの転送を別命令 (LDM) としても、ほとんど速度差は生じない。

—将来CTXBをすべてチップに内蔵して高速化しようという場合には、LDCTXのreservedのオプションやCTXB F Mの機能を利用して仕様を拡張すれば良い。また、LDCTX, STCTX命令では、PC, PSWの転送も行なっていない。これは、次のような理由による。

—一般に、コンテキストスイッチによって切り換える必要があるのは、OSのPCやPSWではなく、ユーザプログラムのPCやPSWである。ところが、ユーザプログラムのPCやPSWは、普通はOS呼び出し時にスタック中に退避されている。そこで、PC, PSWの退避にSP0のスタックを使用するようにしておけば、コンテキストスイッチでSP0を切り換えることにより、PC, PSWも間接的に切り替わる。これを積極的に利用

199

し、CTXBの構造として、SP0から間接参照される部分（スタック）にPC, PSWが置かれるようなものを考えれば、コンテキストスイッチ命令で、PC, PSWの操作（スタック～CTXB間のコピー）をする必要がなくなる。

—SPIを使用した外部割り込みの処理ハンドラの最後で直接コンテキストスイッチを行なう場合には、どうしてもSPIのスタック～CTXB間でPC, PSWの転送が必要になる。しかし、この場合、外部割り込みの中ではコンテキストスイッチを遅延し、外部割り込みから抜ける時にDCEやDIを使ってコンテキストスイッチを行なうようにすれば、DCEやDIでSP0を指定することにより、上記のデータ構造が自然に実現できる。この命令は特権命令である。

【0249】LDCTXによってセットされるPSWのreservedのビット（'—'で表示される）に対して、CTXBから'1'をロードしようとした場合には、予約機能例外（RFE）が発生する。また、UATBなどの制御レジスタのreservedのビット（'—'で表示される）に対して、CTXBから'1'をロードしようとした場合には、単に無視される。これは、LDCによって制御レジスタをセットしようとした場合と同様である。（〈L1〉）仕様のチップでは、AT=*
・予約命令例外

・XX='01'～'11'のとき

・EaA がRn, #imm_data, @SP+, @-SP, 付加モードのとき

・特権命令違反例外

・ring 0以外から実行されたとき

・予約機能例外

・PSWにreservedの値を書き込んだとき

〔ニモニク〕

STCTX

〔命令の機能〕

store context to CTXB

コンテキストのストア

〔命令オプション〕

/LS 論理空間にCTXBをストア

/CS 制御空間にCTXBをストア 〈〈L2〉〉

〔検討中〕

〔命令ビットパターンとアセンブラ表記〕図279に示す

〔フラグ変化〕図280に示す

【0251】〔解説〕プロセッサ中の現在のコンテキストの内容を、CTXBBレジスタで示される領域（CTXB）に退避する。退避の行なわれるレジスタは、MMUの有無やCTXBBFMの内容によって変化するが、SP0～SP3, UATB, CSWなどが含まれる。STCTXで転送されるレジスタの詳細については、付録8を参照のこと。STCTXでは、LDCTXと同様に、

200

*00 （アドレス変換なし）の場合にもUATBの転送が行なわれる。これは、OS内のみで一時的にアドレス変換を中止するというケースが考えられるためである。ただし、AT=00の場合は、/LSを指定してもctxaddrは物理アドレスとして扱われる。LDCTXでUATBを転送しないことを指定するには、CTXBBFMを利用する。LDCTXの現在の仕様では、汎用レジスタの転送を行っていない。しかし、将来仕様が拡張されたり、コンテキスト退避用のメモリをチップに内蔵したりした場合には、LDCTX命令で複数の汎用レジスタのロードまで行なう予定がある。その場合、ctxaddt/EaA!Aで付加モードを許していると、LDMと同様に、命令が途中で中断した場合の命令再実行が難しくなる。したがって、LDCTXのctxaddr/EaA!Aでは付加モードを禁止している。どうしても付加モードの機能を利用したい場合には、MOVAを使って

MOVA @(&@(...)):A, R0

LDCTX @R0

とすることにより、同等の機能が実現できる。

【0250】〔プログラム例外〕

汎用レジスタやPC, PSWの転送は行なわれない。CTXBBの指す空間は/LS, /CSのオプションで指定する。ただし、/CSオプションは、将来コンテキスト退避用のメモリをチップに内蔵した場合にはじめて意味を持つものであり、〈〈L2〉〉となっている。「本発明装置」では、/CSオプションはサポートしない。

「本発明装置」標準のMMUを内蔵したプロセッサでは、STCTX命令でUATB退避が行なわれる。この場合、STCTX/LSが意味のある動作を行なうには、CTXBBがSRを指している必要があるが、CTXBBがSRを指すかURを指すかのチェックは特に行なわない。この命令は特権命令である。STCTXでCTXBに退避される制御レジスタのreservedのビットのうち、'—', '+', '0', '1'がCTXBにセットされる、また、'—', '#', '*'のビットについては、CTXBにセットされる値は不定であり、インプリメント依存である。これらは、STC命令と同様である。（〈〈L1〉〉）仕様のチップでは、AT=00（アドレス変換なし）の場合にもUATBの転送が行なわれる。これは、OS内のみで一時的にアドレス変換を中止するというケースが考えられるためである。ただし、AT=00の場合は、/LSを指定してもCTXBBは物理アドレスとして扱

201

われる。STCTXでUATBを転送しないことを指定するには、CTXBFMを利用する。

【0252】〔プログラム例外〕

- ・予約命令例外
- ・XX='00'以外のとき
- ・P='1'のとき
- ・特権命令違反例外
- ・リング0以外から実行されたとき

【0253】12-16. MMU関連命令

〔モニク〕

ACS chkaddr

〔命令の機能〕

test access rights

アクセス権のチェック

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図281に示す

〔フラグ変化〕図282に示す

〔解説〕chkaddrで指定したアドレスを含むページのATEを調べ、chkaddrがPRNGからアクセス可能かどうかをチェックする。チェックした結果に応じてフラグをセットする。

read 可能 ==> M_flag
write 可能 ==> Z_flag
execute 可能 ==> L_flag

この命令は特権命令ではなく、ユーザが使用することも可能である。例えばring3からPRNG=ring3のアクセス権をチェックすることもできる。したがって、ページアウトなどのOSの管理すべき情報はできるだけ見せないようになっている。ACSの実行のために必要なSection tableやPage tableがページアウトされていた場合には、通常の命令と同じように、ページ不在例外(POE)として命令を再実行する。このほか、ACS命令でATEを参照中には、アドレス変換例外(ATRE)、バスアクセス例外(BAE)の発生する可能性がある。ACS命令でテストするオペランドのサイズは、バイトと考える。つまり、EaAで示されるアドレスの1バイトが、PRNGからアクセス可能かどうかを示す。複数バイトにまたがる領域のチェックを行なう場合は、ソフトウェアで対処する。

【0254】ACSでは、直前のリングからの処理要求に対するアクセス権のチェックを行なう場合に、PRNGがそのまま利用できる。しかし、例えばring3からring2に処理を依頼し、ring2からさらにring1を呼びたいという場合に、ring1でring3からのアクセス権をチェックしたい場合がある。この時、PRNGはring2となっているので、そのままACS命令を使うことはできず、PRNGをring

202

3に書き換えてからACSを実行する必要がある。このような要求に答えるために、PRNGはユーザからも操作可能なPSMに置いている。PRNGは、ACS命令に対するパラメータという意味を持ったフィールドである。ただし、このままではring3からring0の保護情報まで見えることになる。そこで、PRNG<RNGの時は無条件に

L_flag = M_flag = Z_flag = 0

とすることによって、保護情報が見えるのを防ぐ。ACSにおいて、chkaddrが未使用領域(ページ範囲外)であった場合には、R不可、W不可、E不可と同じように、M_flag=0, Z_flag=0, L_flag=0アクセス権なしとして命令を正常終了するものとする。EITとはならない。

【0255】また、AT=00(アドレス変換なし)の場合は、リング保護のチェックが行なわれないので、すべてのアドレスに対してアクセス権をもっていると考える。実際には、バスアクセス例外(BAE)が発生するためにアクセスできない領域もあるわけだが、そのチェックは行なわない。これは、システムバスに起因するアクセスエラーとメモリ保護に起因するアクセスエラーはレベルの違ったものであり、ACSでは後者のみのチェックを行なうと考えるためである。したがって、AT=00の場合、chkaddrが得られた後はいずれの例外も発生せず、L_flag=M_flag=Z_flag=1(アクセス権あり)として命令を終了するものとする。ACS命令は、命令エミュレーションのプログラムで、リング保護レベルのチェックまできちんとエミュレートしたい場合に利用できる命令である。エミュレーションのプログラムは通常ring0に置かれるので、エミュレートされる命令とは異なったリングで実行されるのが普通である。つまり、リング保護のレベルに関しては、エミュレートされるプログラムとエミュレーションプログラムは異なった環境になっている。そこで、エミュレートされる命令のオペランドをアクセスする前に、ACS命令によって、そのオペランドがエミュレートされる命令と同じリング(PRNG)からアクセスできるかどうかをチェックしてやれば、リング保護に関するエミュレーションまできちんと行なうことが可能になる。ACSのchkaddrの実効アドレス計算において、スタックポインタSPの参照があった場合には、PRNGではなく現在リングRNGのスタックが参照される。

【0256】〔プログラム例外〕

・予約命令例外

・EaA がRn, #imm_data, @SP+, @-SPのとき

〔モニク〕

MOVPA srcaddr, dest (「本発明装置」ではサポートしない)

〔命令の機能〕

move physical address

物理アドレスの転送

〔命令オプション〕なし

〔命令ビットパターンとアセンブラ表記〕図283に示す

〔フラッグ変化〕第284に示す

【0257】〔解説〕srcaddrで指定したオペランドの実効アドレス（論理アドレス）を計算し、それを物理アドレスに変換してからdestに転送する。srcaddrで得られた実効アドレスのアドレス変換の方法は通常の命令とは異なり、UATBレジスタではなくR1レジスタをアドレス変換テーブルのベースアドレスとして使用する。これは、OS等から、現在プログラムが走っている論理空間以外の空間に対する操作も行なえるようにするためである。この命令で、高機能命令と同じように固定番号のレジスタを空間指定に使用したのは、高級言語で直接使用することのない命令なので命令の対称性があまり必要ないこと、ビット割り当てからの制約があること、による。MOVPA命令において、srcaddrを得てから、それを物理アドレスに変換するまでの過程でページ不在例外、アドレス変換例外などが発生した場合、そのエラーはフラッグに反映し、EITは起動しない。これはsrcaddrのアドレス変換に使用するSection TableやPage Tableがページアウトされていた場合、また最終段のページ（ページテーブルではない）がページアウトの場合、さらに変換テーブルのエントリ（ATE）のフォーマットにエラー（予約ATEエラー）があった場合などが含まれる。この時、destは変化せず、Vflagがセットされて命令を終了する。また、ページフォールトがあったかどうかはFflagで示される。エラーやページフォールトがなく、命令を正常に終了した場合には、Vflagはクリアされる。この命令は、基本的にはアドレス演算と考えるので、その他のフラッグは無変化である。MOVPA命令のフラッグ変化をまとめると、図285のようになる。なお、STATEでVflag=0、Fflag=1に相当する場合（次の段のページアウト）は、MOVPAではVflag=1、Fflag=1のページアウトの場合に吸収されているので、STATEとMOVPAとではフラッグ変化のパターンが異なっている。srcaddr, de *

・予約命令例外

・+='0' のとき

・W='1' のとき

・BaA がRn, #imm_data, @SP+, @-SPのとき

・EaWIS が#imm_data, @SP+, @-SPのとき

・〈〈L1〉〉機能例外

・MOVPAの正しいビットパターンがデコードされた

*stなどの実効アドレスを得るまでの過程でページフォールトが生じた場合には、通常の命令と同じようにページ不在例外（POE）が起動される。この命令は特権命令である。

【0258】dest/EaWISでは、@-SPのモードを禁止している。これは、エラーやページアウトの発生によってVflagがセットされ、destの転送ができない場合に、destに@-SPが指定されていると命令動作がまぎらわしくなるためである。LDATE, STATE, LDP, STP, LDC, STC, MOVPA命令の中の特殊空間を参照するオペランドにおいて、付加モードによりメモリの間接参照が起こった場合には、特殊空間の方ではなく論理空間（LS）の方を参照する。また、スタックポインタSPの参照があった場合には、PRNGではなく現在リングRNGのスタックが参照される。特殊空間のアドレスという意味を持つのは、最終的に得られた実効アドレスのみである。MOVPA, LDATE, STATE命令で、対象アドレスのMSBが1の場合（SRを示す場合）は、R1ではなくSATBを使ってアドレス変換を行なう。まとめると、図286のようになる。MOVPA, LDATE, STATEでは、アドレス変換のベースレジスタを、UATBの代わりにR1によって指定している。この時、UATBのreservedの部分に対応するR1のビット（'='で表現された2⁴, 2⁵のビット）が'1'でなくても、特にチェックは行なわず、単に無視される。これは、インプリメントの負担を考慮したためである。チェックが行なわれていなくても、R1の2⁴, 2⁵のビットには必ず'0'を入れてもらうように、マニュアル等で指導する必要がある。srcaddrの実効アドレスを得てから、R1を使ってアドレス変換を行ない、その物理アドレスを得るまでの動作は、ATビットには影響されない。つまり、AT=00であっても、AT=01の時と同じようにsrcaddrのアドレス変換が行なわれて物理アドレスが得られる。これは、アドレス変換の前準備としてこの命令を使用することを想定したためである。もちろん、srcaddr, destの実効アドレス計算（間接参照など）やdestへの書き込みは、AT=00の時物理アドレス対象で行なわれる。

【0259】〔プログラム例外〕

とき

〔ニモニック〕

205

LDATE src, dest addr (「本発明装置」ではサポートしない)

〔命令の機能〕

load address translation *

/AS すべての論理空間のTLB をパージする

/SS R0で指定されたLSIDを持つ論理空間のTLB をパージする

/PT PTE操作

/ST STE操作

〔命令ビットパターンとアセンブラ表記〕図287に示す

〔フラッグ変化〕図288に示す

【0260】〔解説〕dest addrで指定したオペランドの実効アドレス（論理アドレス）を計算し、それを物理アドレスに変換する際に使用するアドレス変換テーブルのエントリ（ATE）に対して、srcで得られるデータの転送を行なう。dest addrに対するアドレス変換の方法は通常の命令とは異なり、UATBレジスタではなくR1レジスタをアドレス変換テーブルのベースアドレス（物理アドレス）として使用する。これは、OS等から、現在プログラムが走っている論理空間以外の空間に対する操作も行なえるようにするためである。dest addrのMSBが1の場合（SRを示す場合）は、R1ではなくSATBを使ってアドレス変換を行なう。/PTオプションの場合、/STオプションの場合とも、R1はSection Tableのベースアドレスを指す。結果的に、/PTの場合には2段の間接参照が、/STの場合には1段の間接参照が行なわれる。ATEへのセットが正常に行なわれた場合には、ATE値の変更によって影響を受けるTLBと論理キャッシュのパージが自動的に行なわれる。

【0261】LSIDとは、複数のコンテキスト（プロセスやタスク）のTLBの混在を許す場合に、それを区別する番号である。複数の論理空間の区別が可能なTLBの場合には、/SSオプションを指定することによって、TLB中のLSIDとR0で示されるLSIDが一致したTLBのみをパージすることができる。なお、現在使用中の論理空間に対するLSIDは、LSID制御レジスタに置かれているが、この命令の実行とは直接関係しない。メモリ管理やTLBの構成はインプリメント依存性の強いところなので、この命令を実装する場合にも、必ずしも/SSオプションをインプリメントする必要はない。また、LSIDの機能も必須のものではない。/SSのオプションが用意されているのは、LSIDのあるプロセッサとLSIDのないプロセッサの互換性を取るためである。PSTLBの項を参照のこと。この命令で、高機能命令と同じように固定番号のレジスタを空間指定に使用したのは、高級言語で直接使用するこ

206

*table entry

ATEのロード

〔命令オプション〕

<<L2>>

とのない命令なので命令の対称性があまり必要ないこと、ビット割り当てからの制約があること、による。この命令では、ATE自体のエラーやページアウトなど、いろいろな場合を見分けるため、F__flag, V__flagを使用する。それぞれの場合における動作は、次のようになる。

【0262】1. dest addrのアドレス変換に使用するSection TableやPage Tableのうち、操作する段より上位段のATEにフォーマットエラー（予約 ATEエラー）があった場合
この場合は、操作対象となるATEまで到達できないので、ATEへのセットは行なわれない。V__flag=1, F__flag=0となって命令を終了する。

2. dest addrのアドレス変換に使用するSection TableやPage Tableのうち、操作する段のATEを含むテーブル、あるいはそれより上位段のテーブルがページアウトされていた場合
この場合も、操作対象となるATEまで到達できないので、ATEへのセットは行なわれない。V__flag=1, F__flag=1となって命令を終了する。なお、途中段のATEで、予約ATEエラーと次段のページアウトが同時に起こった場合には、予約ATEエラーの方を優先し、V__flag=1, F__flag=0とする。

3. それ以外の場合

この場合には、srcのデータがATEにセットされ、V__flagは0となる。LDATEによってATEにセットしたデータのPIビットが0の場合には、それより下位の段のページアウトを示すため、F__flag=1となる。また、セットしたデータがATEとして予約ATEエラーを起こすものであった場合には、やはりF__flag=1となる。この二つのケースは、セットしたATEを使ってアドレス変換を行なおうとすると例外が発生するという点で共通している。セットしたATEにエラーがなく、PIビットが1の場合には、F__flag=0となる。LDATE命令のフラッグ変化をまとめると図289のようになる。

【0263】この命令は、基本的にはアドレス演算と考えるので、M__flag, Z__flagなどは変化しな

207

い。また、src, destaddrなどの実効アドレスを得るまでの過程でページフォールトが生じた場合には、通常の命令と同じようにページ不在例外（POE）が起動される。この命令は特権命令である。LDATE／STではPSTLB／STに相当する処理が、LDATE／PTではPSTLB／PTに相当する処理が自動的に行なわれる。LDATE, STATE, LDP, STP, LDC, STC, MOVPA命令の中の特殊空間を参照するオペランドにおいて、付加モードによりメモリの間接参照が起こった場合には、特殊空間の方ではなく論理空間（LS）の方を参照する。また、スタックポインタSPの参照があった場合には、PRNGではなく現在リングRNGのスタックが参照される。特殊空間のアドレスという意味を持つのは、最終的に得られた実効アドレスのみである。MOVPA, LDATE, STATEでは、アドレス変換のベースレジスタを、UATBの代わりにR1によって指定している。この時、UATBのreservedの部分に対応するR1のビット（'='で表現された2⁴, 2⁵のビット）が'1'でなくても、特にチェックは行なわず、単に無視さ

・予約命令例外

- ・!R='11' のとき (!='0' のときは検出しない)
- ・P='1' のとき
- ・ttt='010' ~ '111' のとき
- ・EaR が@-SPのとき
- ・BaA がRn, #imm_data, @SP+, @-SPのとき

・〈〈L1〉〉機能例外

- ・LDATEの正しいビットパターンがデコードされたとき

〔ニモニック〕

STATE srcaddr, dest
「本発明装置」ではサポートしない

〔命令の機能〕

store address translation
table entry

ATEのストア

〔命令オプション〕

／PT PTE操作

／ST STE操作

〔命令ビットパターンとアセンブラ表記〕図290に示す

〔フラッグ変化〕図291に示す

【0265】〔解説〕srcaddrで指定したオペランドの実効アドレス（論理アドレス）を計算し、それを物理アドレスに変換する際に使用するアドレス変換テーブルのエントリ（ATE）を読みだし、destに設定する。srcaddrに対する実効アドレスのアドレス変換の方法は通常の命令とは異なり、UATBレジスタではなくR1レジスタをアドレス変換テーブルのベース

208

*れる。これは、インプリメントの負担を考慮したためである。チェックが行なわれていなくても、R1の2⁴, 2⁵のビットには必ず'0'を入れてもらうように、マニュアル等で指導する必要がある。AT=00でLDATEを実行した場合、srcのフェッチとdestaddrの実効アドレス計算は、他の命令と同様にアドレス変換なしで行なわれる。しかし、LDATEの命令動作そのものは、ATの値に関係しない。すなわち、AT=00であっても、得られたdestaddrの実行アドレスは論理アドレスであると解釈され、それを物理アドレスに変換する際に使用するATEに対して、srcの転送を行なう。これは、アドレス変換の前準備としてこの命令を使用することを想定したためである。AT=00の場合のLDATE, STATE, MOVPAの仕様は、AT=01の場合の仕様との整合性のほか、OSが最初にMMUの動作環境を設定するために利用できるように、また、ユーザプログラムがAT=01、OSがAT=00で動く場合に矛盾なく利用できるように、という意図で決められたものである。

【0264】〔プログラム例外〕

アドレス（物理アドレス）として使用する。これは、OS等から、現在プログラムが走っている論理空間以外の空間に対する操作も行なえるようにするためである。なお、srcaddrのMSBが1の場合（SRを示す場合）は、R1ではなくSATBを使ってアドレス変換を行なう。／PTオプションの場合、／STオプションの場合とも、R1はSectionTableのベースアドレスを指す。結果的に、／PTの場合には2段の間接参照が、／STの場合には1段の間接参照が行なわれる。この命令で、高機能命令と同じように固定番号のレジスタを空間指定に使用したのは、高級言語で直接使用することのない命令なので命令の対称性があまり必要ないこと、ビット割り当てからの制約があること、による。この命令では、ATEの予約ATEエラーやページ不在エラーなど、いろいろな場合を見分けるため、F__flag, V__flagを使用する。それぞれの場合における動作は、次のようになる。

【0266】1. srcaddrのアドレス変換に使用するSectionTableやPageTableのうち、操作する段より上位段のATEに予約ATEエラーがあった場合

この場合は、操作対象となるATEまで到達できないので、ATEの読みだしは行なわれない。V__flag =

209

1. `F_flag=0` となって命令を終了する。
 2. `srcaddr` のアドレス変換に使用する `SectionTable` や `Page Table` のうち、操作する段の ATE を含むテーブルあるいはそれより上位段のテーブルがページアウトされていた場合

この場合も、操作対象となる ATE まで到達できないので、ATE の読みだしは行なわれない。`V_flag=1`, `F_flag=1` となって命令を終了する。なお、途中段の ATE で、予約 ATE エラーと次段のページアウトが同時に起こった場合には、予約 ATE エラーの方を優先し、`V_flag=1`, `F_flag=0` とする。

3. それ以外の場合

この場合には、ATE が読み出されて `dest` にセットされ、`V_flag` は 0 となる。STATE によって読み出された ATE の PI ビットが 0 の場合には、それより下位の段のページアウトを示すため、`F_flag=1` となる。また、読み出された ATE が予約 ATE エラーを起こすものであった場合には、やはり `F_flag=1` となる。この二つのケースは、読み出された ATE を使ってアドレス変換を行なおうとすると例外が発生するという点で共通している。読み出された ATE に予約 ATE エラーがなく、PI ビットが 1 の場合には、`F_flag=0` となる。STATE 命令のフラグ変化をまとめると、図 292 のようになる。なお、フラグ変化の意味を考えると、STATE の `F_flag, or, V_flag` に相当するものが MOVPA の `V_flag` となっており、STATE と MOVPA とではフラグ変化のパターンが異なっている。この命令は、基本的にはアドレス演算と考えるので、`M_flag, Z_flag` などとは変化しない。また、`srcaddr, dest` などの実効アドレスを得るまでの過程でページフォールトが生じた場合には、通常の命令と同じようにページ不在例外 (POE) が起動される。この命令は特権命令である。

【0267】`dest/EaW!S` では、@-SP のモードを禁止している。これは、途中段の予約 ATE エラーやページアウトの発生によって `V_flag` がセットされ、`dest` の転送ができない場合に `dest` に @-SP が指定されていると命令動作がまぎらわしくなるためである。LDATE, STATE, LDP, STP, LDC, STC, MOVPA 命令の中の特殊空間を参照するオペランドにおいて、付加モードによりメモリの間 *

/AS すべての論理空間の TLB をページする

/SS RO で指定された LSID を持つ論理空間の TLB をページする

〔命令ビットパターンとアセンブラ表記〕図 293 に示す

〔フラグ変化〕図 294 に示す

【0269】〔解説〕TLB のページを行なう。TLB

210

*接参照が起こった場合には、特殊空間の方ではなく論理空間 (LS) の方を参照する。また、スタックポインタ SP の参照があった場合には、PRNG ではなく現在リング RNG のスタックが参照される。特殊空間のアドレスという意味を持つのは、最終的に得られた実効アドレスのみである。AT=00 で STATE を実行した場合、`srcaddr` と `dest` の実効アドレス計算は、他の命令と同様にアドレス変換なしで行なわれる。しかし、STATE の命令動作そのものは、AT の値に関係しない。すなわち、AT=00 であっても、得られた `srcaddr` の実効アドレスは論理アドレスであると解釈され、それを物理アドレスに変換する際に使用する ATE を `dest` に転送する。これは、アドレス変換の準備としてこの命令を使用することを想定したためである。MOVPA, LDATE, STATE では、アドレス変換のベースレジスタを、UATB の代わりに R1 によって指定している。この時、UATB の `reserved` の部分に対応する R1 のビット ('=' で表現された 2⁴, 2⁵ のビット) が '1' でなくても、特にチェックは行なわず、単に無視される。これは、インプリメントの負担を考慮したためである。チェックが行なわれていなくても、R1 の 2⁴, 2⁵ のビットには必ず '0' を入れてもらうように、マニュアル等で指導する必要がある。

【0268】〔プログラム例外〕

・予約命令例外

・t='0' のとき

・w='i' のとき

・EaA が Rn, #imm_data, @SP+, @-SP のとき

・EaW!S が #imm_data, @SP+, @-SP のとき

・〈〈L1〉〉機能例外

・STATE の正しいビットパターンがデコードされたとき

〔ニモニック〕

PTLB (「本発明装置」ではサポートしない。

つまり〈〈L2〉〉である。)

〔命令の機能〕

purge TLB.

TLB のページ

〔命令オプション〕

のロックやイネーブルなどの細かい操作を行なうには、制御レジスタを用いる。しかし、TLB に対するページ操作しか行なわない場合には、そのためだけに制御レジスタを追加するのはインプリメントの負担が大きい

211

め、TLBのパージ命令を別に用意している。LSIDとは、複数のコンテキスト（プロセスやタスク）のTLBの混在を許す場合に、それを区別する番号である。／SSオプションの際には、ROにより示されたLSIDを持つ論理空間のTLBのみがパージされる。なお、現在使用中の論理空間に対するLSIDは、LSID制御レジスタに置かれているが、この命令の実行とは直接関係しない。PTLB命令では、特定の論理アドレスのTLBのみをパージする機能はなく、指定した論理空間のすべてのTLBがパージされる。特定の論理アドレスのTLBをパージする場合は、PSTLB命令を使う。ただし、／SSオプションが指定された場合には、指定した論理空間のURのTLBのみがパージされ、SRのパージは一切行なわれない。SR部分のパージを行なう場合には、必ず／ASを使用する必要がある。この命令は特権命令である。メモリ管理やTLBの構成はインプリ

メント依存性の強いところなので、この命令は〈〈L *

/AS すべての論理空間のTLBをパージする

/SS ROで指定されたLSIDを持つ論理空間のTLBをパージする

/PT 論理アドレス全体（2³¹〜2¹²ビット）がprgaddrと一致するエントリをパージする

つまり、PTEの変更時に影響を受ける部分をパージする

/ST 論理アドレスの2³¹〜2²²ビットがprgaddrと一致するエントリをパージする

つまり、STEの変更時に影響を受ける部分をパージする

/AT 論理アドレスの2³¹ビットがprgaddrと一致するエントリをパージする

つまり、UATB. SATBの変更時に影響を受ける部分をパージする

〔命令ビットパターンとアセンブラ表記〕図295に示す

〔フラッグ変化〕図296に示す

【0271】〔解説〕特定の論理アドレスのTLBをパージする。／PTオプションを指定した場合には、対象となる論理空間のTLBのうち、STE〜PTEのインデクスに相当する論理アドレス（すなわち論理アドレス全体）がprgaddrと一致するものをパージする。また、／STオプションを指定した場合には、対象となる論理空間のTLBのうち、STEのインデクスに相当する論理アドレスがprgaddrと一致するものをパージする。／ATオプションを指定した場合には、対象となる論理空間のキャッシュのうち、論理アドレスのMSBがprgaddrと一致するエントリをすべてパージする。LSIDとは、複数のコンテキスト（プロセスやタスク）のTLBの混在を許す場合に、それを区別する番号である。／SSオプションの際には、ROにより示されたLSIDを持つ論理空間のURのTLBのみが

212

*2〉〉となっている。また、この命令を実装する場合にも、必ずしもすべてのオプションをインプリメントする必要はない。LSIDの機能も必須のものではない。PTLBでは、AT=00の場合にも、AT=01の時と同様にパージが実行される。これは、アドレス変換の準備としてPTLB命令を使用することを想定したためである。

【0270】〔プログラム例外〕

・予約命令例外

〔ニモニック〕

PSTLB prgaddr （「本発明装置」ではサポートしない。つまり《L2》である。）

〔命令の機能〕

purge specific TLB

特定のアドレスのTLBのパージ

〔命令オプション〕

パージされる。なお、現在使用中の論理空間に対するLSIDは、LSID制御レジスタに置かれているが、この命令の実行とは直接関係しない。この命令は特権命令である。メモリ管理やTLBの構成はインプリメント依存性の強いところなので、この命令は〈〈L2〉〉となっている。また、この命令を実装する場合にも、必ずしもすべてのオプションをインプリメントする必要はない。LSIDの機能も必須のものではない。

【0272】／AS、／SSオプションは、LSIDの有無に対する互換性を保つために設けてあるオプションである。意味的には、PSTLBの場合に常に／SSのみが指定できればよいが、PSTLBの場合に常に／SS指定とすると、LSIDの有無によって互換性が失われる恐れがある。例えば、最初にLSIDの機能のないプロセッサができると、その上で動くプログラムは、ROにLSIDのセットを行わずにPSTLB命令を実行するものになるだろう。同じプログラムを将来LSIDの機能の有るプロセッサで実行した場合、その時にR

213

0に残っていたゴミによって、全くでたらめのLSIDに対してPSTLBが実行されることになる。これを防ぐためには、オプションを使って、R0をセットしていない場合には/AS指定、将来R0をセットした場合は/SS指定とすればよいわけで、PSTLBにおける/*

/SS/PT

/SS/ST

はすべて有効であり、

/SS はR0によって指定される論理空間のURのTLB をページ

/AS はすべての論理空間に対するTLB のページ、あるいはLSIDの機能の

は使用しない)

となる。

/ASオプションを使えば、LSIDのあるプロセッサでもLSIDのないプロセッサでも共通のプログラムを書けるが、LSIDの機能は生かせないことになる。一方、/SSオプションを使えば、LSIDの機能は生かせるが、LSIDのないプロセッサでは未実装オプションということでエラー(予約命令例外など)になる。

【0273】PTLB, PSTLB命令で/SSオプション※

PTLB/SS R0で指定された論理空間のURをページ

PSTLB/SS/AT @uraddr :uraddrはUR

R0で指定された論理空間のURをページ

PSTLB/SS/AT @sraddr :sraddrはSR /SSでSRを指定したので、何も

しない

SR 全体をページする場合はPSTLB/AS/AT

@sraddr を利用

PSTLB/SS/PT @uraddr :uraddrはUR

R0で指定された論理空間のURの一部をページ

PSTLB/SS/PT @sraddr :sraddrはSR/SS でSRを指定したので、何も

しない

SRの一部をページする場合はPSTLB/AS/PT

@sraddr を利用

PSTLBで/STオプションの実装が難しい場合には、互換性のため機能を縮退してそのまま実行することにし、EITとはしない。具体的には、/STの代わりに/AT相当の動作を行なうことになる。AT=00でPSTLBを実行した場合、prgaddrの実効アドレス計算は、他の命令と同様にアドレス変換なしで行なわれる。しかし、PSTLBの命令動作そのものは、ATの値に関係しない。すなわち、AT=00であつても、得られたprgaddrの実効アドレスは論理アド

50

214

*AS指定はこのような意味を持っている。したがって、PSTLBでは、

/AS/PT

/AS/ST

※ヨンが指定された場合には、指定した論理空間のURのTLBのみがページされ、SRのページは一切行なわれない。SR部分のページを行なう場合には、必ず/ASを使用する必要がある。PTLB, PSTLBで/SSのオプションを指定した場合の動作をまとめると、以下

20 ようになる。

レスであると解釈され、AT=01の時と同様にページが実行される。これは、アドレス変換の前準備としてPSTLB命令を使用することを想定したためである。

【0274】〔プログラム例外〕

・予約命令例外

【0275】付録1. 本発明装置命令セットレファレンス

*: 本発明装置ではサポートしない命令

(データ転送命令)

215	MOV	src, dest	データの移動と符号拡張	216
	MOVU	src, dest	データの移動とゼロ拡張	
	PUSH	src	スタックにプッシュ	
	POP	dest	スタックからポップ	
	STM	reglist, dest	複数レジスタのストア	
	LDM	src, reglist	複数レジスタのロード	
	MOVA	srcaddr, dest	実効アドレスを得る	
	PUSHA	srcaddr	実効アドレスをスタックにプッシュ	
(比較・テスト命令)			* (算術演算命令)	
CMP	src1, src2	比較、符号拡張と比較		
CMPU	src1, src2	ゼロ拡張と比較		
CHK	bound, index, xreg	配列の範囲のチェック		
			*	
	ADD	src, dest	加算、符号拡張と加算	
	ADDU	src, dest	ゼロ拡張と加算	
	ADDX	src, dest	キャリーを含めた加算	
	SUB	src, dest	減算、符号拡張と減算	
	SUBU	src, dest	ゼロ拡張と減算	
	SUBX	src, dest	キャリーを含めた減算	
	MUL	src, dest	乗算	
	MULU	src, dest	符号なし乗算	
	MULX	src, dest, tmp	拡張乗算、サイズが大きくなる	
	DIV	src, dest	除算	
	DIVU	src, dest	符号なし除算	
	DIVX	src, dest, tmp	拡張除算、サイズが小さく剰余を出す	
	REM	src, dest	剰余	
	REMU	src, dest	符号なし除算による剰余	
	NEG	dest	補数演算	
<<L2>>	INDEX	indexsize, subscript, xreg	多次元配列のアドレス計算	
【0276】 (論理演算命令)			SHL	count, dest 論理シフト
AND	src, dest	論理積	SHA	count, dest 算術シフト
OR	src, dest	論理和	ROT	count, dest 回転
XOR	src, dest	排他的論理和	SHXL	dest 拡張左シフト
NOT	dest	全ビット反転	SHXR	dest 拡張右シフト
(シフト命令)			RVBY	src, dest バイト順の逆転
			<<L2>> RVBI	src, dest ビット順の逆転
			(本発明装置ではサポート)	
			(ビット操作命令)	

	217		218
BTST	offset, base	ビットのテスト	*【0277】 (固定長ビットフィールド命令)
BSET	offset, base	ビットのセット	
BCLR	offset, base	ビットのクリア	
BNOT	offset, base	ビットの反転	
BSCH	data, offset	0 または 1 のサーチ	

(1ワード内)

		*	
BFEXT	offset, width, base, dest	ビットフィールドの抽出	(符号付き)
BFEXTU	offset, width, base, dest	ビットフィールドの抽出	(符号なし)
BFINS	src, offset, width, base	ビットフィールドの挿入	(符号付き)
BFINSU	src, offset, width, base	ビットフィールドの挿入	(符号なし)
BFCMP	src, offset, width, base	ビットフィールドの比較	(符号付き)
BFCMPU	src, offset, width, base	ビットフィールドの比較	(符号なし)

(任意長ビットフィールド命令)

BVSCH	0 または 1 のサーチ (任意長ビットフィールド)
BVMAP	ビットマップ演算
BVCPY	ビットマップ転送
BVPAT	パターンとビットマップの演算

(10進演算命令)

* ADDDX	src, dest	BCDの加算
* SUBDX	src, dest	BCDの減算
* PACKss	src, dest	BCDへのパック
* UNPKss	src, dest, adj	BCDからのアンパック

30

SMOV	ストリングのコピー
SCMP	ストリングの比較
SSCH	ストリングのサーチ
SSTR	同一データを繰り返し書き込み (ストリングのフィル)

(キュー操作命令)

【0278】 (ストリング命令)

QINS	entry, queue	ダブルリンクのキューへ挿入
QDEL	queue, dest	ダブルリンクのキューエントリを削除
QSCH		キューのサーチ

(ジャンプ命令)

219			220
BRA	newpc	ジャンプ (PC相対)	
Bcc	newpc	条件ジャンプ (PC相対)	
BSR	newpc	サブルーチンジャンプ (PC相対)	
JMP	newpc	ジャンプ	
JSR	newpc	サブルーチンジャンプ	
ACB	step, xreg, limit, newpc	インデクス値を増加するループ命令	
SCB	step, xreg, limit, newpc	インデクス値を減少するループ命令	
ENTER	local, reglist	スタックフレームの形成、高級言語用サブルーチン ジャンプ	
EXITD	reglist, adjsp	高級言語用サブルーチンリターンとパラメータ解放	
RTS		サブルーチンからのリターン	
NOP		ノーオペレーション	
PIB		命令キャッシュやパイプラインの整合性をとる	

【0279】 (マルチプロセッサ命令)

BSETI	offset, base	ビットのセット (バスをロック)
BCLR1	offset, base	ビットのクリア (バスをロック)
CSI	comp, update, dest	比較とストア (バスをロック)

(制御空間、物理空間操作命令)

LDC	src, dest	制御空間へのロード
STC	src, dest	制御空間からのストア
LDPSB	src	PSB へのロード
LDPSM	src	PSM へのロード
STPSB	dest	PSB からのストア
STPSM	dest	PSM からのストア
LDP	src, dest	物理空間へのロード
STP	src, dest	物理空間からのストア

【0280】 (OS関連命令)

* JRNG	vector	リング間コール
* RRNG		リング間リターン
TRAPA	vector	ソフトウェア割り込み、トラップ
TRAP		条件によるソフトウェア割り込み、トラップ
REIT		EIT 処理 (割り込み、例外処理) からのリターン
WAIT	imask	停止、割り込み待ち
LDCTX	pcbaddr	プロセスコンテキストのロード
STCTX		プロセスコンテキストのロード

(MMU関連命令)

221	ACS	chkaddr	アクセス権のチェック	222
	* MOVPA	srcaddr, dest	物理アドレスの転送	
	* LDATE	src, destaddr	ATE のロード	
	* STATE	srcaddr, dest	ATE のストア	
	<<L2>>* PTLB		TLBのバージ	
	<<L2>>* PSTLB	prgaddr	特定のアドレスの TLBのバージ	

【0281】 (符号付き10進演算命令)

<<L2>>* DCADD	src, dest	符号付きBCD の加算
<<L2>>* DCADDU	src, dest	符号なしBCD の加算
<<L2>>* DCSUB	src, dest	符号付きBCD の減算
<<L2>>* DCSUBU	src, dest	符号なしBCD の減算
<<L2>>* DCX	src, dest	キャリーを含めたBCD の加減算
<<L2>>* DCADJ	src, dest	符号付きBCD の補数
<<L2>>* DCADJU	src, dest	符号なしBCD の補数
<<L2>>* DCADJX	src, dest	キャリーを含めたBCD の補数
<<L2>>* DCCMP	src1, src2	符号付きBCD の比較
<<L2>>* DCCMPU	src1, src2	符号なしBCD の比較
<<L2>>* DCCMPX	src1, src2	キャリーを含めたBCD の比較

【0282】 付録2. 本発明装置のアセンブラ表記について

【0283】 A2-1. 概要

この資料は、命令ニモニック、アドレッシングモードのニモニック、などに関する本発明装置での規定を示したものである。ドキュメントの記述の意味を明確にし、本発明装置に対する理解を深めてもらうことを目的としている。

A2-1-1. このドキュメントにおける記述方法
(...) メタキャラクタを示す。

[A]	Aはあってもなくてもよい。
{A} *	Aの0回以上の繰り返し
{A} +	Aの1回以上の繰り返し
A ::= B C	AはB またはC
A ::= BC	AはB とC をつないだもの

A2-1-2. ニモニック決定の方針

①総称ニモニックとフォーマット別ニモニックを設ける。総称ニモニックは各命令に対応したニモニックであり、短縮形、一般形などフォーマットが複数存在する命令でも、総称ニモニックは一つである。これに対して、フォーマット別ニモニックは、短縮形や一般形などの区別をしたい場合のニモニックである。命令フォーマットを表わす文字を決めておき、総称ニモニックから規則的にフォーマット別ニモニックを作る。ユーザがアセンブラのソースプログラムを書いた場合には、通常総称ニモニックを使う。総称ニモニックに対する最適なフォーマットの選択は、原則としてアセンブラが行なう。

②データタイプ指定子に関して統一的な規則を設ける。データタイプ関係で記述を必要とするものは、演算のためのデータタイプ指定、命令全体でのオペランドサイズ指定、およびオペランド毎のサイズ指定である。これらに関して統一的な規則を設ける。

③ニモニックは、原則としてIEEE Microprocessor Assembly Language Standard (P694) を標準とする。ただし、これには一般的な感覚になじまないと思われるところ、本発明装置のアーキテクチャに合わないところなどがあるので、あくまでも個々の名称を決める際の参考とするだけである。考え方や規則まで完全にIEEEに合わせるわけではない。

④特殊記号の利用はできるだけ避ける。ここで定義するアセンブラでは、できるだけ特殊記号を使用しないという方針にしている。それは、オペランドに数式が来たり、アセンブラを拡張した場合に、その中で使用する記号と競合させないためである。また、文字セットの少ない大型機でも開発を行なうためには、あまり多くの記号を使用するのは望ましくない。特殊記号の利用をできるだけ避けたため、アセンブラの中では括弧を一種類しか使用しておらず、また' ; , ' & 'などが未使用の特殊記号となっている。

【0284】 A2-1-3. アセンブラ命令

本発明装置用アセンブラ言語における一つの命令は、一つのオペレーションニモニックと複数個(0個, 1個を含む)、のオペランドニモニックにより記述される。オペコードニモニックとオペランドニモニックの間は一個

以上の空白文字（スペースまたはタブ）により区切られ、オペランドニモニックどおしの間はコンマ、' に *
 < アセンブラ命令 > ::=

< オペレーション > [< オペランド > { (< オペランド >) }] *

A2-1-4. オペランドの順序

オペランドの順序は命令毎に定まっているが、原則は次のようになる。

移動命令 (MOV)

第一オペランドがソース、第二オペランドがデスティネーションになる。すなわち、

第一オペランド ==> 第二オペランド

これは I E E E 標準と同じである。

2項演算の2オペランド命令 (SUB など)

第一オペランドが2番目のソース、第二オペランドが1番目のソースとデスティネーションになる。すなわち、

第二オペランド, op, 第一オペランド ==> 第二オペ
 < オペレーション > ::=

[< データタイプ >] < 演算操作 > [< バリエーション >]

* { / < オプション > } * { : < フォーマット > } * [< サイズ >]

例:

```
MOV
SMOV/NE.W
MOV.W
MOV:L
MOV:Q.W
```

< データタイプ > 命令の先頭で指定するのは、演算方法に大きな影響を与えるデータタイプ、すなわち、< 演算操作 > に対して直交関係にないデータタイプである。このデータタイプには、ストリング、キュー、ビットフィールドなどが含まれる。データサイズ (整数の8, 16, 32, 64ビット、浮動小数の32, 64ビットなど) の指定は、ここではなく < サイズ > で行なう。また、符号付き、符号なしの指定、およびアドレス演算の指定は、ここではなく < バリエーション > で行なう。

< 演算操作 > 演算そのものの指定を行なう。できる限り I E E E に合わせる。条件ジャンプ命令の条件の指定は本来オプションとするべきであるが、慣例にしたがって基本部分の < 演算操作 > に含める。

< バリエーション > 演算に対する細かい操作や属性の指定を行なう。

< オプション > 命令フォーマット中の数ビットで表現される命令オプションを示す。オプションになるのは、ストリング命令の終了条件、キューのサーチ条件などである。

【0287】< フォーマット > 短縮形、一般形などのフォーマットを指定する。通常は書かなくてもよく、書

※ランド

これは I E E E 標準とは異なるが、多くのプロセッサで用いられている方法であり、なじみやすい。

【0285】A2-2. オペレーションのニモニック

A2-2-1. ニモニック生成規則

I E E E では、演算操作を示す動詞をニモニックの先頭にもってくるという考え方であるが、本発明装置ではさらにその前にデータタイプ指定子を置く。演算操作そのものに対するニモニックは、ほぼ I E E E に合わせる。本発明装置での命令のニモニックは、次のような規則で生成する。

【0286】

[< データタイプ >] < 演算操作 > [< バリエーション >]

* { / < オプション > } * { : < フォーマット > } * [< サイズ >]

30 かない場合は総称ニモニックになる。アセンブラのソースで < フォーマット > を書かずに総称ニモニックを使った場合には、アセンブラで自動的に最適なフォーマットを選ぶ。< フォーマット > を書いた場合にはフォーマット別ニモニックの記述になる。アセンブラのソースでユーザが < フォーマット > を書いた場合には、強制的にそのフォーマットを使うことを示す。< フォーマット > によるフォーマット別ニモニックは、仕様書やマニュアルの記述において、あるいは逆アセンブラなどにおいて、あえて命令フォーマットの区別をしたい場合に用いる。

40 < サイズ > オペランドのサイズを指定する。

< サイズ > を使用する命令は、主として整数を扱う命令と浮動小数を扱う命令である。< サイズ > は < データタイプ > とは異なり、< 演算操作 > に対して直交関係があるのが特徴である。

【0288】A2-2-2. データタイプ

< データタイプ > を表わす文字として、次のようなものがある。なし 整数演算、アドレス演算、雑命令など

225

F	浮動小数
S	ストリング
Q	ダブルリンクによるキュー
B	1ビットのビット操作
BF	固定長ビットフィールドの操作
BV	任意長ビットフィールドの操作

【0289】A2-2-3. 演算操作

原則としてIEEEのニモニックに従う。使用するものは次の通りである。ADD, SUB, MUL, DIV, 10
CMP, NEG, AND, OR, XOR, NOT, L
D, ST, MOV, PUSH, POP, WAIT, NO
P

注意

・MOV, LD, STの使い分け

MOV レジスタ間、メモリ間の転送

LD メモリからレジスタへの転送

ST レジスタからメモリへの転送

LD, STは、方向性を意識する必要のある命令に使用

X	拡張演算	例: ADDX, MULXなど
U	符号なしデータの演算	例: MOVU, ADDU, MULU など
C	制御空間(制御レジスタ)に対する演算	例: LDC, STC
P	物理空間に対する演算	例: LDP, STP
I	バスをロックして行なう処理	例: BSETI, BCLR, CSI
M	複数データの処理	例: LDM, STM
A	アドレス計算	例: MOVA, PUSHA, MOVPA
D	10進演算(符号なし、データチェックなし)	例: ADDDX, SUBDX
	またはスタック上のパラメータを捨てる処理	例: EXITD

【0291】A2-2-5. フォーマット

(フォーマット)は、命令フォーマットの詳細を区別したい場合に用いる。次のような文字を使用する。

Q リテラル短縮形

ビットフィールド命令のスタティック形

ループ命令のリテラル短縮形

例: MOV:Q.W	#3.@abs
BTST:Q.B	#4.@abs
ACB:Q	#1.R1.#5.loop1

40

226

する。

・シフト関係のオペレーションは、左右の指定方法が異なるためにIEEEのニモニックをそのまま使うわけではないが、IEEEの原則を生かしてSHA, SHL, ROTとする。

・ブランチ(条件分岐)命令に関しては、IEEEに従うと'BV'などが別の意味とぶつかること、符号付き整数の比較と符号なし整数の比較の区別をわかりやすくしたいこと、などを考慮したため、条件指定の部分がIEEEには従っていない。

・JMP, JSR, RTSは、ブランチ命令とのバランスからIEEEには従っていない。

・〈バリエーション〉の'X'により拡張演算を表わすことに統一したため、ADDX, SUBX, MULX, DIVXについてもIEEEに従っていない。

【0290】A2-2-4. バリエーション

〈バリエーション〉は、演算に対する属性などを指定するものである。次のような文字を使用する。

227

R

レジスタ-レジスタ間演算の短縮形

ループ命令のレジスタ短縮形

例: AND:R.W R1,R2

MOVA:R.W @ (disp:16,R2),R3

ACB:R #1,R1,R2,loop2

228

L

メモリーレジスタ間演算の短縮形

例: ADD:L.W @abs,R2

MOV:L.W @ (disp,R2),R3

S

レジスタ-メモリー間演算の短縮形 (MOV のみ)

例: MOV:S.W R2,@abs

I

イミディエート短縮形

例: ADD:I.W #100000,@abs2

G

2オペランド命令の一般形

ループ命令の一般形

例: ADD:G.W @abs1,@abs2

ACB:G @abs1,R1,@a

bs2,loop3

E

2オペランド命令の一般形の8ビットイミディエート

例: ADD:E.W #100.B,@abs2

8

newpc が8ビット

例: ACB:G @abs1,R1,@abs2,

loop3:8

16

newpc が16ビット

例: BEQ:G error:16

32

newpc が32ビット

例: BNE:G next:32

64

newpc が64ビット

例: BRA:G loop:64

なお、ここで示した ' : Q ' , ' : G ' といったフォーマット指定は、一つの命令 (総称ニモニック) の中でフォーマットの区別を行ない、フォーマット別ニモニックを作ることが目的のものである。つまり、アセンブラ表記上のフォーマット指定である。一方、命令フォ

ーマット説明で用いた G-format, E-format といったフォーマットの方は、命令全体の中でフォーマットの説明を行なうことが目的のものである。したがって、例えば同じ ' : G ' であっても ' MOVA : G ' であれば MOVA 命令の一般形なので G-format であり、' MOV : G ' であれば MOV 命令の一般形なので G-format ということになる。

【0292】A2-2-6. サイズ

IEEEでは64ビット整数まで考慮されていないので、扱うデータサイズはどうしてもIEEEと異なったものになる。

整数の場合

4通りのサイズが対称的にサポートされている点、オペランド側でもデータタイプが指定できる点、が特徴であ

229

る。オペレーション側にもオペランド側にも同じものを書くので、' . ' により区切っている。〈サイズ〉として、次のような文字を使用する。

B	Byte	8ビット長の整数
H	Halfword	16ビット長の整数
W	Word	32ビット長の整数
L	Longword	64ビット長の整数

'L' は本発明装置32では使用できない。

浮動小数点の場合

別に定める。

【0293】A2-3. オペランドのニモニック

オペランドには、汎用のアドレッシングモードまたはそのサブセットが利用できるもの（一般オペランドと呼ぶ）と、命令に応じた特殊な指定をするもの（特殊オペランドと呼ぶ）とがある。特殊オペランドについては、命令毎にフォーマットを定める。特殊オペランドをとる命令は、

BRA, Bcc, BSR, ACB, SCB (newpc オペランド)
LDM, STM (reglist オペランド)

などである。

〈オペランド〉 ::=

〈一般オペランド〉

| | 〈特殊オペランド〉

一般オペランドに関しては、本発明装置において、オペランド毎にデータサイズを指定できる点が特色であり、アセンブラにおける一般オペランドの記述方法にもその*

【本発明装置アドレッシングモードの表記原則】

@Aまたは@ (A) アドレスA のメモリ内容を参照

mem[A]

@ (A, B, C, ...) A, B, C. . を加算して、加算結果のアドレスのメモリ内容を参照

mem[A + B + C + ...]

本発明装置における' () ' は、間接参照などの特別な意味は持たない。一般の数式と同じように、結合の順序を示すに過ぎない。したがって、@Aと@ (A) は全く同じ意味ということになる。以下で説明するシンタックスにおいて' (. .) ' が入る場合であっても、' (. .) ' 内に一つの項しかなければ、' (. .) ' を省略しても構わない。

* ような能力を持たせている。また、オペランドに対しては総称ニモニックとフォーマット別ニモニックを設けている。一般オペランドのニモニックは、実際のオペランドの値（実効アドレス）と付加モードフォーマットの区別、およびサイズから成る。

〈一般オペランド〉 ::=

〈オペランド値〉[:〈付加モード指定〉][.〈サイズ〉]

【0294】A2-3-1. アドレッシングモード表記

10 の原則

従来のプロセッサでは、アドレッシングモードの数があり多くなかったため、それぞれのモードを個別に考え、別々の記号を割り当てておけばよかった。また、表記法と実際のアドレッシングのオペレーションがうまく対応していない場合も見られた。例えば、あるプロセッサではレジスタ相対間接のアドレッシングモードをdisp(Rn)で表現する場合があるがこれはオペレーションとしてはmem[disp+Rn]であり、dispの部分とRnの部分の扱いが対称的ではない。これだけで使っている場合には問題は生じないが、これを組み合わせると複雑なモードを作ってしまう場合には矛盾の起ることがある。本発明装置では付加モードといった機能があるため、統一的、規則的なアドレッシングの表記を行なわないと混乱を招く。そこで、本発明装置では実際のオペレーションとその表記との関係について原則を設け、それに基づいて、付加モードまで一貫したアドレッシングモードの表記を行なうこととした。アドレッシングは、基本的には加算と間接参照の繰り返しである。したがって、この二つのオペレーションに対する表記法が決まればよい。本発明装置の表記原則をまとめると次のようになる。

ば、' (. .) ' を省略しても構わない。

【0295】従来のプロセッサでは' (. .) ' によって間接参照を意味する場合があり、これがある程度慣用的な表記法となっている。しかし、このような表記法では以下のような点で誤解を生みやすい。

231

232

例:

慣用的表記法	オペランド値
Rn	Rn
(Rn)	mem[Rn]
abs	mem[abs]またはabs
(abs)	mem[mem[[abs]]]またはmem[abs]

うまく対応がとれない場合が生ずる。

本発明装置において、間接参照を必ず'@'によって表現しているのは、こういった理由による。イミディエート、スタック操作のアドレッシングモード、インデックスのスケールリングなどの処理はこの原則に入らないので、原則を参考にしながらそれぞれ別に表記法を定める。

【0296】A2-3-2. 付加モードの指定

〈付加モード指定〉 ::= A ! ! N

'A'の指定は、付加モードのフォーマットを使うということを特に強調したい場合に付け加える。また、'*

例:

@(disp, PC):A	常にPC相対付加モードを使用 dispが32ビット以下であっても付加モードを使用
@(disp, PC):N	常にPC相対間接のモードを使用 dispが64ビットであるとエラー
@(disp, PC)	dispが32ビットであればPC相対間接のモードを使用 dispが64ビットであればPC相対付加モードを使用

【0297】A2-3-3. サイズ

〈サイズ〉は、オペランドの演算サイズを示すものであり、オペレーションのニモニックに示されたサイズと組みになって実際のサイズ指定を行なう。サイズ指定の文字は、オペレーションに使われるものと同じである。オペランド中の〈サイズ〉と、オペレーション中の〈サイズ〉の関係は、原則として次のようになる。

・オペレーション中に〈サイズ〉の指定があった場合には、その〈サイズ〉が全部のオペランドのデフォルトの※

例:

MOV.W	@src, @dest	src, destともW(WORD)型
MOV.W	@src.B, @dest	srcはB(BYTE)型、destはW(WORD)型
MOV	@src.B, @dest.W	srcはB(BYTE)型、destはW(WORD)型

【0298】A2-3-4. オペランド値

以下の項では、一般オペランドに関してアドレッシングモード別にそのアセンブラ表記を示す。〈イミディエート値〉、〈絶対アドレス〉などの内容としては、数値、変数名、数式などが書けるが、そのシンタックスは別に定める。〈フォーマット〉は、アドレッシングモードの

10 * N'の指定は、付加モードを使用しないということを特に強調したい場合に付け加える。これらの指定は、フォーマット別ニモニックに相当するものである。': N', ': A'とも書かれていない場合は、そのアドレッシングが付加モード以外の短いモードで実現できるかどうかをアセンブラが判断し、実現できればそのモードを使う。付加モードでないと実現できなければ、付加モードを使う。

※サイズとして有効になる。ただし、サイズの指定ができないオペランド、イミディエートのオペランド、特殊な意味を持つオペランドの場合はこの限りではない。

・オペランド中に〈サイズ〉の指定があった場合には、それがそのオペランドのサイズになる。オペレーション中で異なるサイズが指定されていても、オペランドで指定された〈サイズ〉の方が優先される。

・オペランド中で〈サイズ〉の指定を行ない、それが利用できないサイズであった場合はエラーとなる。

フォーマット選択を明示したい場合を書く。主としてアドレッシングモードの拡張部のサイズを指定するために使用する。通常は書かなくてもよく、書かない場合はアセンブラで自動的に最適なフォーマット(サイズ)を選ぶ。〈フォーマット〉によるフォーマット別ニモニックは、仕様書やマニュアルの記述において、あるいは逆ア

233

234

センブラなどにおいて、あえてアドレッシング部のフォーマットの区別をしたい場合に用いる。
 <フォーマット> ::= 4 | 16 | 32 | 64

4 4ビット長のアドレッシング修飾部
 16 16ビット長のアドレッシング拡張部
 @ (disp:16, Rn), abs:16など
 32 32ビット長のアドレッシング拡張部
 @ (disp:32, Rn), abs:32など
 64 64ビット長のアドレッシング拡張部
 abs:64など

【0299】<フォーマット>により指定されるのは、あくまでも命令フォーマット自体のサイズである。一方、<サイズ>により指定されるのは、演算されるオペランドのサイズである。イミディエートモードの場合を除けば、両者は全く異なるものである。

例:

MOV R0.W, @addr:16.W

*メモリに転送する。絶対アドレッシングが用いられている。' : 16 ' は ' addr ' を16ビットで表現することを示す。したがって、' addr ' の範囲は \$ f f f f 8 0 0 0 ~ \$ 0 0 0 0 7 f f f となる。一方、' . W ' は演算をWORD (32ビット)で行なうことを示す。つまり、この命令により4バイトのデータが転送される。<レジスタ番号>に入るのは、汎用レジスタの20 モニックである。

この命令では、R0の内容を ' addr ' で示されるメモリ
 <レジスタ番号> ::=

R0 | | R1 | | R2 | | R3 | | R4 | |
 R5 | | R6 | | R7 | | R8 | | R9 | | R10 | |
 R11 | | R12 | | R13 | | R14 | | R15 | |
 FP | | SP

FPとR14は全く同義、SPとR15は全く同義である。

【0300】A2-3-4-1. レジスタ直接 30
 オペランド=Rn
 <オペランド値> ::=
 <レジスタ番号>

例:

R1

A2-3-4-2. レジスタ間接

オペランド = mem[disp16 + Rn]
 mem[disp32 + Rn]

<オペランド値> ::=

@ (<ディスプレイメント>[:<フォーマット>], <レジスタ番号>)

<フォーマット> ::= 16 | 32

例:

@ (disp:16, R5)

オペランド = mem[Rn]

<オペランド値> ::=

@ <レジスタ番号>

例:

@R2

【0301】A2-3-4-3. レジスタ相対間接

235

A 2-3-4-4. リテラルとイミディエート
オペランド = imm_data

〈オペランド値〉 ::=

#〈リテラル値〉

〈オペランド値〉 ::=

#〈イミディエート値〉

リテラルの命令フォーマットを使用するということを明
例:

ADD:Q.W #1.R0 リテラルのフォーマットを使用 (2 バイト)

ADD:I.W #1.R0 イミディエート形のフォーマットを使用 (6 バイト)
ソースオペランド '1' は32ビットで表現

ADD:L.W #1.R0 短縮形のフォーマットを使用 (6 バイト)
ソースオペランドとして32ビット
イミディエートを指定

ADD:G.W #1.B.R0
一般形のフォーマットを使用 (6 バイト)
ソースオペランドとして8ビット
イミディエートを指定
16ビットフィールドの下位8ビットを用いて '1' を表
現。'1' は32ビットに符号拡張される。

ADD:B.W #1.R0
一般形8ビットイミディエートのフォーマットを使用
(4 バイト) '1' は32ビットに符号拡張される。

ADD:G.W #1.R0
#1にはサイズの指定がなく、しかも:Gフォーマットな
のでサイズの自由度が残されている。したがって、自
動的に最小のサイズが選択され、

ADD:G #1.B.R0.W (6 バイト命令)
と等価になる。

ADD:G #1.W.R0.W (8 バイト命令)

236

示したい場合は、オペレーションのニモニック中で示
す。イミディエートの場合には、拡張部のサイズがオペ
ランドのサイズとして決まるため、〈フォーマット〉と
〈サイズ〉が同じ意味になる。アセンブラとしては、
〈フォーマット〉としても〈サイズ〉としてもその大き
さを指定することができる。

【0302】なお、イミディエートのオペランドでオペ
ランド側にサイズ指定がなく、しかも命令機能の上でサ
イズの自由度がある場合には、自動的に最小のサイズが

10 選択されるものとする。

237

238

になるのではない。

ADD:G.W #1:16.R0

これは、〈サイズ〉ではなく〈フォーマット〉によって命令を選択した例であり、

ADD:G.W #1.H.R0

と等価になる。

総称ニモニックで単に

ADD.W #1.R0

と書くと、最もコードの短い

ADD:Q.W #1.R0

が選択される。

【0303】また、命令によっては、サイズが一つに固定されているわけではないが、実質的にほとんど一つのサイズでのみ使用されるものがある。そのようなものについては、特にオペランド側に〈サイズ〉がついていない*

*い限り、命令によって定められるデフォルトサイズを適用する。これは、〈オペレーション〉のニモニックが全体のオペランドにかかるという原則に対しては、例外となる。

【例】

ビット操作命令のアクセスサイズ(BB指定)は、8bit(.B)がデフォルト。
 T.H..W は<<L2>>..L は<<LX>>固定長ビットフィールド操作命令のレジスタサイズ(X指定)は、32bit(.W)がデフォルト。
 .H..B は使用不可、.Lは<<LX>>

BTST.W R0, @addr = BTST R0.W, @addr.B

【0304】A2-3-4-6. PC相対間接

A2-3-4-5. アブソリュート

オペランド = mem[abs16]

mem[abs32]

mem[abs64]

30

〈オペランド値〉 ::=

@〈絶対アドレス〉[:〈フォーマット〉]

〈フォーマット〉 ::= 16 | 32 | 64

例:

@abs:32

40

239

240

オペランド = mem[disp16 + PC]
 mem[disp32 + PC]

< オペランド値 > ::=

@([< ディスプレースメント >[:< フォーマット >]]).PC)

< フォーマット > ::= 16 | 32

例:

@(disp, PC)

A2-3-4-7. スタックポップ

オペランド = mem[SP++]

< オペランド値 > ::=

@SP+

例:

@SP+

【0305】A2-3-4-8. スタックプッシュ
 オペランド = mem[--SP]

*【0306】A2-3-4-9. FP相対間接

< オペランド値 > ::=

@-SP

例:

@-SP

*30

オペランド = mem[disp4 + FP]

< オペランド値 > ::=

@([< ディスプレースメント >[:< フォーマット >]], < レジスタ番号 >)

< フォーマット > ::= 4

< レジスタ番号 > ::= FP | R14

例:

@(disp4:4, FP)

このアドレッシングモードでは、ビットパターン中に指定されたdisp値を4倍して実際のディスプレースメントとするが、アセンブラでの表記に使う値は、4倍した後のものである。〈フォーマット〉を指定しない場合には、アセンブラでの表記がレジスタ相対間接のモードと同じになるため、アセンブラによって最適なモードが

選択される。つまり、@(disp, Rn)と書かれたオペランドでは、RnがR14またはFPであり、かつdispが-32〜31の4の倍数の時にはFP相対間接のモードが選択され、それ以外の場合にはレジスタ相対間接のモードが選択されるわけである。

50 【0307】A2-3-4-10. SP相対間接

241
オペランド = mem[disp4 + SP]

242

< オペランド値 > ::=

@([< ディスプレースメント >[:< フォーマット >]],< レジスタ番号 >)

< フォーマット > ::= 4

< レジスタ番号 > ::= SP | R15

例:

@(disp4:4, SP)

このアドレッシングモードでは、ビットパターン中に指定された disp 値を4倍して実際のディスプレースメントとするが、アセンブラでの表記に使う値は、4倍した後のものである。

A2-3-5. 付加モード

付加モードについても、機能面の要求を示す総称ニモニックと、フォーマットやビットパターンを記号化したフ *
ベースモードまたはそれまでの付加モード tmp 値

=> ディスプレースメント

=> インデクス

を原則とする。こうすると、実効アドレス計算の流れが左から右への単純な形になり、先の段の付加モードに必要な情報が先の方に、後の段の付加モードに必要な情報が後ろの方に集まる。すなわち、総称ニモニックの表記の順序が、付加モードの機械語ビットパターンの順序と同じになる。したがって、フォーマット別ニモニックや※

:B その部分の処理をベースモードにより実現することを表わす

:A その部分までの処理を一段の付加モードにより実現することを表わす

:N その部分の処理は次の段の付加モード(' :A' の指定のある部分)でまとめて実現することを表わす

なお、「その部分の処理」とは、フォーマット指定文字がディスプレースメントやレジスタに付いている場合にはその値の加算処理を、フォーマット指定文字が閉じ括弧')'についている場合は間接参照の処理を意味する。また、' :A' で「その部分までの処理」とあるのは、その' :A' の部分の処理と、それより左側で' :N' の付いた部分(前の' :A' または' :B' との間で' :N' の付いた部分)の処理を合わせて行なうことを示す。

・フォーマットをすべて指定した場合には、' :A' の個数が付加モードの段数になる。また、通常は一回の' :A' が一段の間接参照に対応する。ただし、複数のインデクスレジスタを加算する場合(間接参照がなく

*フォーマット別ニモニックを設ける。

【0308】[総称ニモニックに関して]

・@または@(. . .)により間接参照を表わし、(. . ., . . ., . . .)によりアドレスの加算を表わすという原則はそのままである。

・表記の順序は、

※実際の機械語の付加モードとの対応がよく、アセンブラも簡単になり、理解しやすくなる。

【0309】[フォーマット別ニモニックに関して]

・フォーマット指定用として、つぎの3つの文字を導入することにより、機械語のビットパターンと1対1に対応した表記ができるようにする。

30

でも' :A' が必要)、最後の段で二重間接を行なう場合(二段の間接参照でも一回の' :A' でよい)は例外である。

・フォーマットの表記のない場合には、総称ニモニックとして表記した処理を実現できるような付加モードが自動的に選択される。また、実際の付加モードでは実現できないフォーマットをフォーマット別ニモニックで指定した場合は、エラーとなる。さらに、フォーマット指定ニモニックからフォーマット指定文字を取り去ると、そのまま総称ニモニックになる。このような点は、フォーマット別ニモニックの一般的な原則と同じである。

【0310】[フォーマット一般に関して]

・複数のアドレス加算がない場合、@(. . .)の括

243

弧は書かなくてもよい。よって、例えば 付加モードで実現される3重間接参照の@ (@ (@ (R1))) は@@@R1と書いてもよい。これは、付加モード以外のアドレッシングにも適用される原則であり、一種のシンタックスシュガーと言える。

・インデックスのスケール値は、IEEEでは' : ' : W' などサイズ指定文字を使用しているが、
例：

@(offset, PC)

mem[offset + PC]

総称ニモニック。offsetが32ビットに入ればPC相対間接モード

で実現し、32ビットに入らなければ付加モードで実現する。

@(offset, PC):N

mem[offset + PC]

必ずPC相対間接モードで実現し、付加モードにはしない。

本発明装置64で、offsetが32ビットに入らない時はエラーとなる。

@(offset[:N], PC[:N]):A

mem[offset + PC]

必ず付加モードで実現する。ベースモードの指定に相当する部分がないので、

絶対付加モード

+ 付加モード E[=10, disp= offset, index=PC, scale=1

になる。

@(PC[:B], offset[:N]):[A]

mem[offset + PC]

必ず

PC相対付加モード

+ 付加モード E[=10, disp=offset, index=0, scale=*

で実現する。

@(@(@R3[:B], base1[:N], R4*4[:N]):[A], base2[:N], R5[*1][:N]):[N])[

:A]

mem[mem[mem[R3 + base1 + R4*4] + base2 + R5]]

R3相対付加モード

+ 付加モード E[=01, disp=base1, index=R4, scale=4

+ 付加モード E[=11, disp=base2, index=R5, scale=1

@(R3[:B], base1[:N], R4*4[:A], R5*2[:N]):[A]

mem[R3 + base1 + R4*4 + R5*2]

R3相対付加モード

+ 付加モード E[=00, disp=base1, index=R4, scale=4

244

将来スケール値にもっと大きな値を入れることも考えられるので、ここは従来通りスケール値の数字を直接書くようにする。また、スケール指定する文字もIEEEの' : 'ではなく、' * 'を使用する。これは、' : 'をフォーマット指定の目的で別に使用しているためである。


```

245      + 付加モード   EI=10, disp=base2, index=R5, scale=2
      0(R3[:B], base1:A, R4*4:A):A
      mem[R3 + base1 + R4*4]
      R3相対付加モード
      + 付加モード   EI=00, disp=base1, index=0, scale=*
      + 付加モード   EI=00, disp=0, index=R4, scale=4
      + 付加モード   EI=10, disp=0, index=0, scale=*
246

```

これは一段の付加モードで実現可能であるにもかかわらず 10 * にまとめておく。ただし、括弧を省略する略記法、および、フォーマットを指定してわざわざ三段の付加モード び各部分を区切るコンマ', ' についてのシンタックスにしている例である。付加モードのシンタックスを以下 * は、ここには含めていない。

オペランド = mem[mem[...] + disp + Rn * Scale1 + Rm * Scale2...]

【0311】

```

<一般オペランド> ::=
    <オペランド値>[:N][.<サイズ>]
    | | <付加モードオペランド値>[:<サイズ>]

<付加モードオペランド値> ::=
    0(<付加モード中間値>, [<disp値>[:N]], [<インデクス値>[:N]])[:A]
    EI=10 に対応
    | | 0(0(<付加モード中間値>, [<disp値>[:N]], [<インデクス値>[:N]])[:N])[:A]
    EI=11 に対応
    これは付加モード最終段を表わす。

<付加モード中間値> ::=
    <付加モード中間値>, <disp 値>[:A]
    | | <付加モード中間値>, [<disp値>[:N]], <インデクス値>[:A]
    EI=00 に対応
    | | 0(<付加モード中間値>, [<disp値>[:N]], [<インデクス値>[:N]])[:A]
    ]
    EI=01 に対応

```

これは途中にある一段の付加モードを表わす。

【0312】

```

<付加モード中間値> ::=
    [0[:B]]      絶対付加モードに対応
    | | <レジスタ番号>[:B] レジスタ相対付加モードに対応
    | | PC[:B]      PC 相対付加モードに対応

```

これはベースモード（レジスタ相対付加モード、PC相 対付加モード、絶対付加モードの区別）を表わす。

247

248

<disp 値> ::=
 <ディスプレイメント>[:<フォーマット>]
 D, dddd フィールドに対応

<フォーマット> ::= 4 | 16 | 32 | 64

<インデクス値> ::=
 <レジスタ番号>[:<サイズ>]['*'<スケール値>]
 | | PC[:<サイズ>]['*'<スケール値>]
 S, M, R_x, XX フィールドに対応

<サイズ> ::= W | L

<スケール値> ::= 1 | 2 | 4 | 8

'*' は、アスタリスク '*' を文字として使うことを示す。「繰り返し」を表わすメタ的な意味はもたない。

<インデクス>の<サイズ>は、インデクスレジスタの有効なデータサイズである。本発明装置64で、W'を指定した場合には、レジスタの下位32ビットが64ビットに符号拡張される。<インデクス>の<スケール>を省略した場合には、1が仮定される。

【0313】A2-3-6. 特殊オペランド
 一般のアドレッシングモード以外の方法で指定されるオペランド(特殊オペランド)に関しては、次のようなシンタックスとする。なお、各部分を区切るコンマ', 'についてのシンタックスは、ここには含めていない。
 reglist (LDM, STM, ENTER, EXITD命令)

<レジスタ番号>または<レジスタ番号>-<レジスタ番号>を', 'で区切って並べ、'(. . .)'でくくったもの

<特殊オペランド> ::=
 ((<連続レジスタ番号>)) *

<連続レジスタ番号> ::=

<レジスタ番号>

その番号のレジスタを指定

| | <レジスタ番号>-<レジスタ番号>

間の番号のレジスタをすべて指定

例:

ENTER.W #10.()

LDM.W @block.(SP)

STM.W (R1, R3, R9-R13, FP).@-SP

newpc (BRA, Bcc, BSR, ACB, SCB 命令)

アドレッシング方法は、PC相対モードのみである。オ 50

ペランドとしては、単にジャンプ先のラベルのみを書く。この場合は、アセンブラによって、その命令の先頭アドレスとジャンプ先のアドレスとの差がnewpcのビットパターンとして設定され、命令実行時にそのラベルの場所へジャンプできるようになる。

<特殊オペランド> ::=

<行き先ラベル>

【0314】

例:

BEQ nextaddr nextaddr にジャンプ

ACB.B #1, R1, @limit, loopaddr

loopaddr にジャンプ

30 BRA, Bcc, BSR, ACB, SCB命令は、出現頻度が高い、特殊なアドレッシング(PC相対のみ)である、慣用的に行き先のラベルをそのまま書ける方がよい、といった理由から、<行き先ラベル>を書くだけで自動的に<行き先ラベル>のアドレスと、これらの命令の置かれたアドレスとの差がディスプレイメントに設定されるようになっている。すべてのオペランド表記のうちで、レジスタ以外のシンボル名が先頭に'#'も'@'もなく現われるのは、この<行き先ラベル>に限られる。したがって、例えば

40 BRA label

は

JMP @label-\$, PC)

と同じ意味を表わすことになる。なお、'\$'は、この記号を含む命令(この場合はJMP命令)の先頭アドレスを表わす。

249

adj (UNPKss 命令)

頭に '#' を付ける。

< 特殊オペランド > ::=

| | #< 補正值> 値がそのまま設定される

例:

UNPKBW @src.@dest.#H' 23302330

vector (TRAPA 命令)

頭に '#' を付ける。

< 特殊オペランド > ::=

| | #< ベクトル> 値がそのまま設定される

例:

TRAPA #1

【0315】その他

・ビットフィールド命令などのリテラル指定は、短縮形のリテラル指定と同じように # (リテラル値) で表わす。

・CHK, INDEX, ACB, SCB ビットフィールド命令などのレジスタ指定は、一般のアドレッシングにおけるレジスタ直接モードと同じように、(レジスタ番号) で表わす。

【0316】A2-4. フォーマット別ニモニックと総称ニモニック

「総称ニモニック」と「フォーマット別ニモニック」は、本発明装置アセンブラの特徴の一つである。従来のプロセッサでも一部の命令について似たような考え方は見られたが (68020 の MOV と MOVQ など)、本発明装置では両方のニモニックを完全に体系化し、オペレーションだけではなくオペランドの記述にも同じ考え方を取り入れた点に特色がある。フォーマット別ニモニックと総称ニモニックの間には、次のような関係がある。

・インプリメントやフォーマットから来るいろいろな制約をユーザに押し付けたくないのが総称ニモニックであり、総称ニモニックを書く限りアセンブラが最適なコードを選ぶ。同じ機能を持ち、同じようにフラッグのセットされる命令は、できる限り一つの総称ニモニックに統合する。

・フォーマット別ニモニックは、機械語のビットパターンと 1 対 1 に対応するものである。フォーマット別ニモニックが変わっても、それはオブジェクトサイズや実行サイクル数にのみ影響し、ユーザから見た命令機能はフ

250

ラッグ変化まで含めて全く同じである。この点でフォーマット指定子とサイズ指定子は根本的に異なる。サイズ指定子の場合は、演算サイズが変わるとユーザから見た命令機能も変化する。条件ジャンプ命令では B R A 1 a b e l : 3 2 のようなフォーマット指定子を使っているのに、加算命令の場合は、ADD s r c . B , d e s t . W のようなサイズ指定子を使っているのは、このためである。

・通常ユーザが使うのは、「総称」の方である。「フォーマット別」は、仕様書でのフォーマットの説明、逆アセンブラなどの特殊な用途でしか使わない。したがって、場合によっては多少冗長な感じになることもあるが、使用目的を考えると特に問題はない。ユーザが主に使用するのは、あくまでも総称ニモニックである。また、「総称」と「フォーマット別」は両極端の表記法に過ぎず、一部のみのフォーマットを指定する中間的な記法も存在する。例えば、@ (o f f s e t , P C) を付加モードで書きたいが、付加モードの各段のフォーマットはどうでもよいという場合には、

@ (o f f s e t , P C) : A

と記述する。「フォーマット別」と言っても、どうしてもフォーマット指定の必要な部分のみ指定を行なえばよく、実際にはそれほど長い記述にはならないのが普通である。

・「フォーマット別」から「総称」への変換は、' : X ' を取ることにより機械的にできるようになっている。また、逆の変換も、フォーマットが許される範囲で適当に ' : X ' を付ければ済むようになっている。オペランドの順序は変化しない。フォーマット別ニモニックとして、記号を変えたり、順序を変えたりする方法も考えられるが、そうすると総称ニモニックとの関係がスムーズではなくなるため、そのような方法はとっていない。(いろいろな場合分けが必要になり、かえってわかりにくくなる。拡張性もよくない。)

また、前述の @ (o f f s e t , P C) : A のように、一部のフォーマットのみ指定したい場合には、「フォーマット別」か「総称」かの区別が統一的行なえる方が望ましい。

・結局、ユーザからの要求を受けるインタフェースが総称ニモニックであり、機械語からくる制約を受けるインタフェースがフォーマット別ニモニックである。両者の調整をとるのが、' : X ' のフォーマット指定文字およびアセンブラである。

・フォーマット別ニモニックと総称ニモニックを併用することの欠点は、アセンブラが複雑化することである。しかし、ユーザがフォーマットのことまで心配するよりは、アセンブラで処理できることはアセンブラで処理する方がよいと考えており、そのためにアセンブラが多少複雑になるのはやむを得ない。

・機械やフラッグ変化の異なる命令は、たとえ命令のビ

251

ットパターンが似ていたとしても別の総称ニモニックになる。

【0317】以上のような理由から、「フォーマット別」の方は、多少記述が長くなっても、「フォーマット別ニモニックであること」や「どのフォーマットを使うかということ」がはっきりわかるようにした方がよいと考えている。フォーマットを表わす部分をすべて「X'」に統一しているのは、このためである。なお、シンタックス中で「[...]」とある部分は省略可能であるが、省略するかどうか全体で統一されている必要はない。例えば、ある「[...]」を省略し、別の「[...]」を残してもよい。

【0318】A2-5. 言語としてのアセンブラ
これまでに述べたアセンブラ表記は、機械語のビットパターンに命令としてのニモニックを与えるための表記法であり、アセンブリ言語の核となる部分である。本発明装置では、ここまですべて「(LO)」仕様とする。しかし、アセンブラを一つの言語として見た場合には、ニモニック以外にも次のような項目を規定する必要がある。これらの項目については、本発明装置のアーキテクチャと矛盾を起こさない範囲で、できるだけIEEEに合わせるように標準化する。

- ・大文字と小文字の使用をどうするか・シンボルの文字数は何文字までにするか
- ・シンボルに式が書けるかどうか、シンタックスはどうなるか
- ・ラベルの形式はどうか(ラベルの後に「:」をつけるかどうか)
- ・2進数、8進数、10進数、16進数の表記法はどうか
- ・コメントの表記法はどうか
- ・ストリングの記述形式
- ・特殊キャラクタ(改行文字「 $\backslash n$ 」など)の表現形式
- ・細かいシンタックスや使用できる文字
- ・アセンブラ擬似命令
- ・マクロ

このうち、2進数、8進数、10進数、16進数の表記法については、IEEEでは以下のような形式になっている。

B' ~	2進数	例: B' 00010010 =	H' 12
Q' ~	8進数	例: Q' 22 =	H' 12
D' ~	10進数	例: D' 18 =	H' 12
H' ~	16進数		

本仕様書でも、16進数を表わすために「H' ~」を、2進数を表わすために「B' ~」を使用している。

【0319】A2-5-1. 大文字と小文字について *

@(@([0:B.] label1-\$[:N].PC[:N]))[:A].label2-\$[:N].PC[:N])[:A]

これは

50

mem[mem[displ + PC] + disp2 + PC]

252

* IEEEでは特に決まっていない。本発明装置では、ニモニックや予約名に対しては、大文字と小文字を同等に扱う。すなわち、この資料で大文字になっている部分に対して、小文字を使用しても構わない。ただしユーザが定義した一般の変数においては、大文字と小文字を区別する方を標準とする。

A2-5-2. シンボル値

〈ディスプレイメント〉、〈リテラル値〉、〈イミディエート値〉、〈絶対アドレス〉などの項目(総称して〈シンボル値〉と呼ぶ)には、定数、ラベルを含む四則演算式が書けるものとする。式の中では、優先順位を変えるために「()」を使用することができる。ただし、未確定の値(外部名や後で定義されるラベルなど)を含む式に関しては、正しいリロケーションができるように、演算式の形を制限しても構わない。さらに、式の中では現在注目している命令のアドレスを示す値として、「\$」を使用することができる。PC相対間接モードの表記は

@(disp, PC)

であり、dispの値が直接ディスプレイメントに設定される。しかし、PC相対でリロケータブルなプログラムを書く場合には、オペランドのアドレスそのものをdispとして設定するのではなく、オペランドのアドレスとこの命令のあるアドレスとの差をdisp値として設定する必要がある。この目的で、「\$」を使用することができる。すなわち、(オペランド-\$)をdisp値として設定すればよい。

【0320】

「\$」を使ったプログラム例

<<アドレス>>

H' 00FE	...
H' 0100	MOV.B #1, @(loc-\$:16, PC)
H' 0104	MOV.B #2, @(8:16, PC)
H' 0108	...
H' 010C	...
	...
H' 0180	loc: ...

40 アドレスH' 0100のMOV.B命令の第二オペランド@(loc-\$:16, PC)では、実際のdispのビットパターンに設定される値がH' 0180-H' 0100=H' 0080となる。この命令により、アドレスH' 0180のlocに1がセットされる。一方、H' 0104のMOV.B命令では、アドレスH' 0104+8=H' 010Cに2がセットされる。

【0321】付加モードと「\$」を使ったオペランドの

表記

を表わす。ただし、
disp1 はlabel1の指すアドレスと現在アドレスとの差
disp2 はlabel2の指すアドレスと現在アドレスとの差

であり付加モードの拡張部は
絶対付加モード

+ 付加モード E[=01, disp=disp1, index=PC, scale=1
+ 付加モード E[=10, disp=disp2, index=PC, scale=1

という構成になる。このモードは、リロケートブルなテ
ーブル (case文用のジャンプテーブルなど) をプロ
グラム領域に置く場合に使用できる。一段目のPC相対
間接

mem[disp1 + PC]

は、case文用のテーブル参照をリロケートブルにする
ために用いられる。また、二段目のPC相対間接

mem[mem[...] + disp2 + PC]

は、ジャンプ先アドレスの決定をリロケートブルに行な
うために用いられる。

【0322】付録3. 本発明装置メモリ管理方式概要

本発明装置を組み込み用などの用途に用いることを考
え、命令セットは本発明装置になっているが、メモリ
管理のハードウェア (MMU) は持たないといったバー
ジョンのチップも考えられる。したがって、本発明装置
のメモリ管理機構は、必ずサポートの必要な〈〈L
0〉〉仕様ではなく、標準的な仕様の揭示のみを行なう
〈〈LA〉〉仕様となっている。以下では、〈〈L

*A) 〉仕様としての本発明装置標準メモリ管理方式を説
明する。

【0323】A3-1. メモリ管理方式の選択と〈〈L IR〉〉仕様

本発明装置では、ハードウェアによるアドレス変換とメ
モリ管理方式 (以下MMUと呼ぶ) の標準仕様が、

〈〈LA〉〉仕様として用意されている。しかし、本発
明装置にITRONやμBTRONを実装するような場
合には、MMUが不要となることが多い。また、MMU
を使用する用途であっても、ページテーブルなどMMU
関係の実行環境の設定が終わるまでは、アドレス変換な
しで命令を実行する必要がある。そこで、本発明装置で
は、MMU機構を使用しているかどうか、アドレス変換
を行なっているかどうか、を示すフィールドをPSW内
に設け、このフィールドを書き換えることにより、アド
レス変換やメモリ保護の有無を指定できるようにしてい
る。このフィールドをAT (Address Translation) フィールドと呼ぶ。ATはPSSのb
it6~bit7に配置されている。ATをPSW内に
設けたことにより、LDCTX等によるコンテキストス
イッチや、EIT処理の起動、REIT命令によるタク
サ処理ハンドラからのリターンの際にも、アドレス変換
の有無を切り換えることが可能である。ATフィールド
の意味を以下に示す。

【0324】

【表2】

AT	意 味
00	アドレス変換、メモリ保護なし
01	本発明装置標準のアドレス変換とメモリ保護〈〈LA〉〉
10	アドレス変換なし、アドレスのみを使った簡単なメモリ 保護〈〈L1R〉〉 (アドレスのMSBによるメモリ領域の区別、2リング)
11	reserved

【0325】このうち、〈〈LA〉〉仕様の標準メモリ
管理を実装した本発明装置ではAT=00, 01が利用
可能、〈〈L1R〉〉本発明装置ではAT=00, 10
が利用可能となる。〈〈L1R〉〉仕様のAT=10の
場合、MMUがないのでページ毎のメモリ保護は行なえ
ないが、〈〈LA〉〉の4リングを縮退してリング0、
リング3のみを有効とし、アドレスによって簡単なメモ
リ保護を行なう。アドレスのMSB=1の領域 (〈〈L
A〉〉でいうSR) はリング0からアクセス可能、リン
グ3からアクセス禁止の領域であり、通常はここにOS
を置く。一方、アドレスのMSB=0の領域 (〈〈L
A〉〉でいうUR) はリング0からもリング3からもア
クセス可能な領域であり、通常はここにユーザプログラ

ムを置く。MMUがないのでユーザプログラム間のメモ
リ保護はできないが、ユーザプログラムからOSを保護
することは可能である。AT=00 (アドレス変換な
し) の場合は、メモリアクセスに対するリング保護のチ
ェックは行なわれない。したがって、ページ不在例外
(POE)、アドレス変換例外 (ATRE) は発生しな
い。ただし、AT=00の場合も特権命令のチェックは
行なわれる。AT=00の時の動作については、〈〈L
1〉〉と〈〈L1R〉〉で全く同じになることが望まし
い。しかし、LDATなどの命令では、〈〈L1〉〉
ならばMMU環境の設定という意味で実用的な命令であ
るのに対して、〈〈L1R〉〉では全く意味を持たな
い。また、PTLBなどの命令も〈〈L1〉〉のAT=

255

00ならば一応意味を持っているが、〈〈L1R〉〉ではTLBそのものがないので全く意味を持たない。したがって、〈〈L1R〉〉仕様ではこういったMMU関係の命令を実装しないことにする。〈〈L1R〉〉でこういった命令を実行しようとした場合は、ATの値にかかわらず予約命令例外(RIE)となる。

【0326】A3-2. 本発明装置のメモリ管理方式
本発明装置は〈〈L1R〉〉仕様のチップである。本発明装置のATフィールドの意味を図297に示す。

【0327】A3-3. 本発明装置のI/O空間アクセスに関して

IOMASK, IOADDRで示されるI/O空間に対する命令フェッチ及びメモリ間接アドレッシング・モードによるオペランドフェッチはアドレス変換例外となる。I/O空間に対するアクセスで、メモリ間接アドレッシングの場合はアクセス動作は一切行なわれない。しかし、命令フェッチの場合にはアクセス動作が行なわれる。そのため、外部のI/Oデバイスはバスアクセスタイプ(BAT)信号をみて命令フェッチであれば応答しないようにする必要がある。I/O空間は通常リング0の領域におかれるためリング3からのアクセスはリング保護違反が検出されることになると考える。リング保護違反の場合は高速に検出可能なためメモリアccessは行なわれない。また、I/O空間とI/Oでない空間をまたぐようなアクセスが行われた場合、アドレス変換例外を起こすが、その場合の再実行動作を保証できない。

【0328】A3-4. メモリ管理の目的と概要

本発明装置では、以下のような目的を達成するために、ハードウェアによるメモリ管理機構(MMU)を導入している。

- ・命令再実行と仮想記憶方式のサポートにより、実装されている物理メモリの量を越える大きさの論理空間を提供する。

- ・多重論理空間の機能の導入により、コンテキスト(タスクやプロセス)間の独立性を維持し、プログラムを作りやすくする。

- ・リング保護の機能の導入により、OSや共有データとユーザプログラムやユーザデータとの間でメモリ保護を行なう。

本発明装置では、以上のような機能を提供するために、毎回のメモリアccessでページング方式によるアドレス変換を行なう。アドレス変換前のアドレス(論理アドレス)の作る空間を、Logical Spaceの意味でLSと予備、アドレス変換後のアドレス(物理アドレス)の作る空間を、Physical Spaceの意味でPSと呼ぶ。ページングの場合には、メモリアccessを高速化するために、TLB(Translation Lookaside Buffer)と呼ばれるアドレス変換対の記憶バッファを導入することが一般的である。しかし、TLBはメモリアccessを高速化するため

256

のインプリメント上の手段と考えられるため、本発明装置のMMU仕様としてはTLBに関する規定は行なわない。本資料でも、TLBに関する説明は行なっていない。また、コンテキストの切り換えによるTLBのページを減らすため、〈〈L2〉〉として論理空間識別子(LSID)の機能を導入することができる。LSIDは、コンテキスト毎にユニークに与えられた番号であり、LSIDまで含めてTLBの論理アドレス比較を行なうようにすれば、コンテキストを切り換える際にもすべてのTLBをページする必要がなくなる。しかし、LSIDの機能についてもインプリメント依存性が強いいため、本発明装置のMMU仕様としては、LSIDの詳細機能やビット数の規定を行なっていない。本発明装置の仕様では、LSIDを示す制御レジスタのアドレス割り当てと、LSIDを実装した場合のTLBやキャッシュの整合性確保に関する説明のみが行なわれている。

【0329】A3-4-1. ページング

本発明装置のアドレス変換は、ページングを基本としている。ページサイズは4KBとして本発明装置全体で統一されている。これによって、TLBの構造などのある程度限定することができ、メモリ管理機構を内蔵できるチャンスが大きくなる。また、基本的なパラメータを固定化すれば、それにチューニングすることで性能向上が期待できる。アドレス変換に際し、本発明装置32の論理アドレスは図298のように分割され、これによって2段階のページングを行なう。32ビットの論理アドレスは、4GBの論理空間を作る。Rビット(論理アドレスのMSB)により、4GBの論理空間は2GBのUnshared Region(UR)とShared Region(SR)に分かれる。おのおののRegionは、SXフィールドにより4MBずつのSectionに分かれる。さらに、おのおののSectionは、PXフィールドによりRKBずつのPageに分かれる。したがって、2段階のページングのうち上位段のページテーブルは、SXをインデクスとして、下位段のページテーブルのベースアドレスを引き出すものとなる。これをSection Table(ST)と呼び、その一つのテーブルエントリをそのエントリをSTEと呼ぶ。また、2段階のページングのうち下位段のページテーブルは、PXをインデクスとして、物理ページのベースアドレスを引き出すものとなる。これをPage Table(PT)と呼び、その一つのテーブルエントリをそのエントリをPTEと呼ぶ。一つのSTEの変更により一つのSectionが、一つのPTEの変更により一つのPageが影響を受けることになる。PTE, STEの総称名としてATE(Address Translation table Entry)という名称を用いる。STのベースアドレスは、URの場合UATB, SRの場合SATBという制御レジスタによって示される。UATBまたはSATBの変更により、

一つのRegion (UR又はSR) が影響を受ける。
以上述べた関係をまとめたものを図299に示す。

【0330】A3-4-2. 多重論理空間

本発明装置では、論理アドレスのMSBによって、共通空間 (Shared Region) と個別空間 (Unshared Region) が区別されている。それぞれのRegionに対して、アドレス変換のテーブルベースレジスタUATB, SATBが存在するが、このうちUATBのみはコンテキスト毎に切り換わるようになっており、これによって多重論理空間を実現する。すなわち、UR (論理アドレスH' 00000000~H' 7ffffff) ではコンテキスト毎に別々の物理空間 (物理メモリ) が割り当てられるのに対して、SR (論理アドレスH' 80000000~H' fffffff) ではコンテキスト間で共通の物理空間 (物理メモリ) が割り当てられる。Shared Regionは主に割り込み処理ハンドラやOSが使用し、Unshared Regionは主にユーザプログラムが使用するものであるが、ユーザデータでもタスクやプロセス間で共用するものはSRを使用する場合があるし、OSの管理するデータでもタスクやプロセス毎に持つ必要のあるものはURを使用する場合がある。多重論理空間の機能を利用すれば、同一の論理アドレスから複数のプログラムを同時に実行することができるので、実行時にオブジェクトコードのリロケーションをする必要がなくなる。また、他のタスクやプロセスのURを直接参照することはできないので、プログラム間のメモリ保護にも役立つ。なお、UR, SRの区別と後述のメモリのリング保護の機能とは、直接の関連はない。すなわち、ring 3からSRを参照できないとか、PCがURにある間はSRを参照できないとかといった制限は、UR, SR自体の機能には含まれていない。このようなメモリのアクセス制限を行ないたい場合には、メモリのリング保護の機能を利用し、STE, PTEに適当な保護コードを設定しておかなければならない。

【0331】〔多重論理空間の構成〕図300において、Unshared Region/Shared Regionの切り替えは、論理アドレスのMSBによって行なわれる。

・Unshared Regionでは、UATBレジスタによってアドレス変換される。

・Shared Regionでは、SATBレジスタによってアドレス変換される。

・UATBレジスタのみコンテキスト毎に切り換わるため、Unshared Regionでは、それぞれのコンテキストが別々の論理空間を持つことができる。アドレス変換が2段のページングとなっているため、異なるコンテキストの2つのSTEが同一のPage Tableを指すことにより、URの中での共用セクション、共有ページを設けることもできる。なお、URとSRの

境界にまたがるようなプログラムやデータについては、以下のように考える。

・64ビット拡張時には、現在連続しているURとSRの境界が不連続になるため、H' 7ffffffから先に伸びているプログラムは、64ビット拡張時に使用できなくなる。これは、本発明装置32で、H' 7ffffffの次のアドレスをH' 00000000と考えてもH' 80000000と考えても同じである。したがって、URとSRの両方にまたがるようなメモリアクセスや、UR~SRで連続するような命令のフェッチを行なうべきではない。

・しかしながら、URとSRにまたがっているかどうかのチェックを毎行行なうのは、インプリメント上の負担が大きいため、URとSRの両方にまたがるアクセスがあったとしてもEITとはしない。この場合、Regionとは関係なく、アドレスがリニアアドレスであるという考え方を生かし、H' 7ffffffの次はH' 80000000、H' fffffffの次はH' 00000000のアドレスをアクセスするものとする。ただし、前項でも述べたように、この仕様を利用するようなプログラムを書くべきではない。PSW, UATB, SATBに対するLDC命令や、LDATE, LDCTX命令による論理空間の切り換えでは、現在実行中のプログラム領域のアドレス変換に対しても影響を与えることができる。したがって、使い方によっては次の命令から全く別の場所に飛んでしまうというケースも生じる。これはプログラムの責任で処理しなければならない。具体的には、ATビットの変更を行なう場合はV=R領域を利用したり、LDCTXを実行する場合はSR (Shared Region) を利用したりすることになる。

【0332】A3-4-3. リング保護

本発明装置のメモリ保護方式は、4レベルのリング保護である。保護情報は、論理アドレスやUR, SRの区別には関係なくページ毎に指定することができる。リング保護では、PSW中に示される現在リング番号(RNG)と、アクセスすべき論理アドレスのSTE, PTEに含まれている保護コードとの関係によって、アクセスが可能かどうかが決まる。RNGが小さい値であるほど、すなわち内側のリングであるほど強いアクセス権を持っており、RNG=3が最もアクセス権が弱い。rng 1 < rng 2とした場合、RNG=rng 2でアクセスできるページは必ずRNG=rng 1でもアクセスできることになる。

【0333】A3-5. MMU関係制御レジスタ

MMUに関係する制御レジスタについて説明する。

・PSW

別項参照。MMUに関連するのは、RNG, ATのフィールドである。

・LSID

259

別項参照。このレジスタの有無と有効なビット数は、インプリメントに依存する。

・UATB (Unshared region Address Translation table Base)

STB: Section Table Base

Section Table の物理ベースアドレス

=: '0' にreserved

ユーザ向けのマニュアルでは、'0' を書き込みように指導しておく

。ただし、'1' を書き込んでも無視される。

読みだし時は値不定である。

D: Direction

Section Table のサイズが小さい場合のSection Table の方向

D=0 Section Table の下位アドレス側が有効

D=1 Section Table の上位アドレス側が有効

D とLLのフィールドは、小規模な用途で、有効な論理アドレスの範囲を制限し、同時にSection Table のサイズを縮小するために使用するものである。

LL: Length

Section Table のサイズ

LL=00 1/2 サイズ 512 エントリ 2KBテーブル 2GB Region

LL=01 1/4 サイズ 256 エントリ 1KBテーブル 1GB Region

LL=10 1/16サイズ 64 エントリ 256Bテーブル 256MB Region

LL=11 1/64サイズ 16 エントリ 64Bテーブル 64MB Region

PI: Page In

PI=0 Section Table が物理アドレスに存在しない。

この場合には、STB はハードウェア的な意味を持たないので、OSで自由に使用することができる。ただし、PI=0でもD, LL のフィールドは有効である。D, LL のフィールドは、アドレス変換例外(ATRE)の未使用領域参照エラーか、ページ不在例外(POB)かを区別するために使用され、例外の検出は前者が優先される。

つまり、PI=0の時にこのレジスタを使ってアドレス変換を行おうとした場合、D, LLのフィールドにより未使用領域参照エラーが検出されればアドレス変換例外(ATRE)が発生し、そうでなければページ不在例外(POB)が発生する。

PI=1 Section Table が物理アドレス上に存在する。

PI=1であれば、このレジスタを使ってアドレス変換を行うことが可能である。ただし、D, LL のフィールドにより未使用領域参照エラーが検出された場合には、アドレス変換例外(ATRE)が発生する。

260

図301に示す。

・SATB (Shared region Address Translation table Base)

図302に示す。

261

【0334】SATB, UATB中のPIビットは、正確にはPage InではなくSection Table Inの意味を表わす。また、後述するSTE中のPIビットは、Page InではなくPage Table Inの意味を表わす。しかし、あえて区別する必要もないと思われるので、いずれも同じ'PI'という名称を使用する。広義の「ページ」は、「ページテーブル」や「セクションテーブル」、つまりディスクへの追出しの単位となるものをすべて含むことになり、ページテーブルやセクションテーブルの不在にも「ページ不在例外」という例外の名称をそのまま適用する。Section Table, Page Table自体をページアウトすることも可能であるが、ページインされているテーブルについては、テーブルのベースアドレス(STB, PTB)は物理アドレスを表わすものとする。すなわち、ページテーブルは論理空間ではなく物理*

Xは有効なビット(0/1)

- は無効なビット(0)

'>>'はシフトを表わす

STB
+)>>20 B' XXXX XXXX XXXX XXXX XXXX XXXX XX-
XXXX XX -

【0336】ここで、'X'の重なっているビット、すなわちSTBの2⁶~2¹¹のビットの扱いが問題となる。以下のような案が考えられる。

①STBの2⁶~2¹¹のビットは'0'でなければならないものとする。インプリメント上は、STBの2⁷~2³¹とSXを20ビット右にシフトしたものとを連結することによって、STEのアドレスを算出することができる。(アラインメント強制)

②STBの2⁶~2¹¹のビットが0でなくてもよいが、2¹¹までの範囲でSTEの実効アドレスがラップアラウンドする。インプリメント上は、重なりのある5ビットのみの加算を行ない、加算により生じた桁上りは無視することになる。(ラップアラウンド)

③STBの2⁶~2¹¹のビットが0でなくてもよく、Section Tableに対して2⁶より上位のアラインメントは全く強制しない。インプリメント上は、まず重なりのある5ビットの加算を行ない、桁上りを生じた場合には最上位桁まで桁上りを伝搬させる必要がある。つまり、2⁶より上位のビットについてすべて加算を行なう必要がある。(アラインメント自由)

【0337】現在③が本発明装置の仕様となっている。LL=01, 10の場合、またPTBとPXのアラインメントの場合も同じように③の仕様とする。DとLLの機能は、小規模な用途でSection Tableの領域を節約するために設けられているものである。DとLLの指定が変わると、Section Tableが小さくなるため、SXの取り得る値に制限ができる。

262

*空間に置かれていると考えることができる。また、Section TableやPage Tableが4Kのフルサイズでない場合にST, PTのページアウトを許すと、64B, 256B, 1KB, 2KB単位でページ不在となるケースも生じる。つまり、UATB, SATB, STEでのPIビットは、必ずしも4K単位でのページ不在を指すとは限らないので、注意する必要がある。セクションテーブル、ページテーブルの大きさが可変になっているが、テーブルが最小(1/64)の大きさでない時には、STBとSXの有効なビットに重なりが生じる。例えば、UATB, SATBでD=0, LL=00とした場合、以下のような計算によりSTEの実効アドレスを算出する必要がある。

【0335】

【数17】

D, LLのそれぞれの値に対して、許されているSXの値は図303のようになる。D=1, LL=00のreservedの部分は、ソフトウェアで使用してはいけない。マニュアルにもreservedであることを明記する必要がある。ただし、この値を実際にUATBやSTABに設定した場合は、D=0, LL=00と同じ動作をする。つまり、LL=00の場合にはDは無視される。論理アドレスとして、上の表に当てはまらない値を指定した場合には、アドレス変更例外(ATRE)の未使用領域参照エラーが発生する。ページ不在例外(POE)とアドレス変更例外(ATRE)が同時に発生した場合には、アドレス変更例外(ATRE)が優先される。例えば、UATBのD=0, LL=10, PI=0で論理アドレスH'40000000をアクセスした場合には、POEではなくATREとなる。DとLLの指定によるテーブル領域の節約のようすを図示すると、図304のようになる。この時、使用する論理アドレスの範囲が狭いため、はじめの数個のSTEのみが利用できるという①の例であれば、図305のようによって、section tableの大きさを節約することができる。又、スタック領域やOS用の領域などで、終わりの数個のSTEのみが利用できれば済むという②の例であれば、図306のようによって、section tableの大きさを節約することができる。

※D=1の場合、STBはsection tableの有効部分の先頭アドレス(SXとしては途中の値に対応する)を指すのではなく、SX=0に対応するSTE

263

264

(実際には存在しない)の置かれるべきアドレスを指す。STE中のPTBの場合も同様である。

【0338】A3-6. STEとPTE

STEとPTEは、メモリ上に置かれるアドレスの変換
PTB: Page Table Base

Page Table の物理ベースアドレス

W: Write可能

W=1 書き込みのアクセス権はPTB の保護コードに従う

W=0 PTE の保護コードにかかわらず、全リングから書き込み禁止

違反した場合はアドレス変換例外(ATRE)のリング保護違反エラーが発生

E: Execute可能

E=1 実行のアクセス権は PTEの保護コードに関わらず、全リングから実行禁止

違反した場合はアドレス変換例外(ATRE)のリング保護違反エラーが発生

D: Direction

Page Table のサイズの小さい場合のPage Tableの方向

D=0 Page Tableの下位アドレス側が有効

D=1 Page Tableの上位アドレス側が有効

LL: Length

Page Table のサイズ

LL=00 フルサイズ 1024エントリ 4KBテーブル 4MB Section

LL=01 1/4 サイズ 256エントリ 1KBテーブル 1MB Region

LL=10 1/16サイズ 64エントリ 256Bテーブル 256KB Region

LL=11 1/64サイズ 16エントリ 64Bテーブル 64KB Region

PI: Page In

PI=0 Page Tableが物理アドレス上に存在しない。

この場合には、PTB はハードウェア的な意味を持たないので、OSに自由に使用することができる。ただし、PI=0でもD, LLのフィールドは有効である。D, LLのフィールドは、アドレス変換例外(ATRE)の未使用領域参照エラーやリング保護違反エラーか、ページ不在例外(POE)かを区別するために利用され、例外の検出は前者が優先される。

つまり、PI=0の時にこのSTE を使ってアドレス変換を行なおうとした場合、D, LLのフィールドにより未使用領域参照エラーやリング保護違反エラーが検出されればアドレス変換例外(ATRE)が発生し、そうでなければページ不在例外(POE)が発生する。

用のディスクリプタである。STEとPTEのフォーマットについて説明する。

・STE (Section Table Entry)

図307に示す。

【0339】

265

PI=1 Page Tableが物理アドレス上に存在する。

PI=1であれば、このSTEを使ってアドレス変換を行なうことが可能である。ただし、D, LLのフィールドにより未使用領域参照エラーやリング保護違反エラーが検出された場合には、アドレス変換例外(ATRE)が発生する。

266

STBとSXの時と同様に、PTBとPXの場合にも、PTBのアラインメントは自由とする。すなわち、LL≠11の時には、PTBとPXの有効なビットに重なりが生じる場合があるが、重なった部分のアドレスについては最上位桁まで加算を行なう。DとLLの機能は、小規模な用途でPage Tableの領域を節約するために設けられているものである。DとLLの指定が変わると、Page Tableが小さくなるため、PXの取り得る値に制限ができる。D, LLのそれぞれの値に対して、許されているPXの値は図308のようになる。論理アドレスとして、上に当てはまらない値を指定した場合には、アドレス変換例外(ATRE)の未使用領域参照エラーが発生する。D=1, LL=00の部分に対しては、W, Eが特別な意味を持つ。D=1, LL=00のSTEを使ってメモリアクセスを行なおうとした場合には、次のような動作をする。この時、SXの値は関係しない。また、PTBのフィールドとEビットは、ハードウェアで使用しないので、OSから自由に利用することができる。

【0340】図309において

‘*’はソフトウェアで自由に使用してよいビットである。ハードウェア的にはこのビットが無視される。このビットは、reservedを示す‘=’とは異なり、将来の使用拡張でも使用しないことがはっきりしているビットである。‘=’と‘*’は、現在の使用におけるハードウェア的な動作は同じであるが、将来の拡張のために仕様書上きの扱いが異なっている。なお、未使用領域参照エラーと予約ATEエラーの区別は、アドレス変換例外(ATRE)が起動された際にスタックに積まれるエラーコードによって行われる。これらのエラーによるアドレス変換例外の起動は、STE, PTEの値を設定した時に検出されるのではなく、値を使用する時(アドレス変換の時、つまりメモリアクセスを行った時)に検出される。D=1, LL=00, W=0の機能は、一

つのsection全体を未使用領域としたい場合に利用する機能である。この場合、このSTEに対するPage Tableは無い。LL=00の機能を利用して小さいサイズのSection Table, Page Tableを使用する場合でも、論理アドレス中のSX, PXの位置は変わらない。したがって、複数のSTEから小さいサイズのPage Tableを使用する場合には、有効な論理アドレスが飛び飛びの値をとることになる。この様子を図310に示す。

【0341】このように、小さいサイズのPage Tableを使った場合には、有効な論理アドレスが連続領域とはならないことがある。しかし、STEを1エントリしか使用しないような小規模な応用のため、あるいは、長さが半端になった論理空間の最後の部分のテーブルについて、「テーブル領域の節約」を行なうためにLL≠00の機能が用意されているのだと考えると、LL≠00の時にアドレスが連続しなくても、特に問題はない。STEでは、D, LLによるアドレス変換例外(ATRE)の未使用領域参照エラー、W=0, E=0によるアドレス変換例外(ATRE)のリング保護違反エラー、PI=0によるページ不在例外(POE)が同時に発生する可能性があるが、例外の検出順序はこの順とする。つまり、まず未使用領域かどうかのチェックを行ない、次にアクセス権のチェックを行ない、最後にページ不在のチェックを行なう。これは、PTEの場合も同様である。したがって、ページ不在の時でも、リング保護関係の情報は有効である。ただし、この例外検出順序は、一つの段(STEまたはPTE)の中での例外検出順序であり、これよりもさらにテーブルを引く順序が優先する。つまり、さたで発生する例外よりもSTEで発生する例外の方が優先される。

【0342】・PTE (Page Table Entry) 図311に示す。

267

268

PFN: Page Frame Number

対応するページの先頭物理アドレス
4KB が単位になる。

** : DSが自由に利用できるビット
ハードウェア的には意味を持たない

R: Referenced

このページが参照されたとき、1 にセットされる。

M: Modified

このページの一部が変更されたとき、1 にセットされる。

RL: Read Level

RL=00 リング0 のみから読みだし可能

RL=01 リング0 ～1 から読みだし可能

RL=10 リング0 ～2 から読みだし可能

RL=11 リング0 ～3 から読みだし可能

違反した場合はアドレス変換例外(ATRE)のリング保護違反エラーを発生

T: Type

T=00 書き込み禁止、実行禁止

T=01 書き込み禁止、実行可能

T=10 書き込み可能、実行禁止

T=11 書き込み可能、実行可能

「書き込み可能」の場合は、 $\min(RL, AL)$ のリングから書き込み可能、それより外側からは書き込み禁止となる。

「実行可能」の場合は、 $\max(RL, AL)$ のリングから実行可能、それより外側からは実行禁止となる。

違反した場合はアドレス変換例外(ATRB)のリング保護違反エラーを発生

【0343】

269

270

AL: Access Level for indicated type

(T ≠ 00の場合)

AL=00 リング0 のみからアクセス (書き込み、実行) 可能

AL=01 リング0 ~1 からアクセス (書き込み、実行) 可能

AL=10 リング0 ~2 からアクセス (書き込み、実行) 可能

AL=11 リング0 ~3 からアクセス (書き込み、実行) 可能

(T=00の場合)

AL=00 リング0 ~RLから読みだし可能

書き込み、実行は禁止 (T, RL の本来の意味で使用)。

AL=01 アドレス変換例外 (ATRE) の未使用領域参照エラーを発生

AL=10 (reserved)

AL=11 (reserved)

T=00の場合は、ALによるアクセス可能リングの指定が必要ないので、ALを別の意味に使っている。このうち、AL=01の指定による未使用領域参照エラー発生の機能は、一つのPage全体を未使用領域としたい場合に利用する機能である。この場合、このPTBに対する物理ページは存在しな

271
い。

272

NC: Non Cachable

NC=1 キャッシュへの取り込みの禁止を指定

1/0 レジスタやVRAMの領域などで、キャッシュへの取り込みやメモリアクセスの順序の変更が許されないページの場合に、このビットをセットする。

NC=0 通常のページであるたとを指定

PI: Page In

PI=0 Pageが物理アドレス上に存在しない。

この場合には、PFN はハードウェア的な意味を持たないので、OSに自由に使用することができる。ただし、PI=0でもRL, T, ALなどのフィールドは有効である。これらのフィールドは、アドレス変換例外(ATRE)の未使用領域参照エラーやリング保護違反エラーか、ページ不在例外(POE)かを区別するために利用され、例外の検出は前者が優先される。

つまり、PI=0の時にこのPTEを使ってアドレス変換を行おうとした場合、RL, T, ALなどのフィールドにより未使用領域参照エラーやリング保護違反エラーが検出されればアドレス変換例外(ATRE)が発生し、そうでなければページ不在例外(POE)が発生する。

PI=1 Page Tが物理アドレス上に存在する。

PI=1であれば、このPTEを使ってアドレス変換を行なうことが可能である。

ただし、RL, T, ALなどのフィールドにより未使用領域参照エラーやリング保護違反エラーが検出された場合には、アドレス変換例外(ATRE)が発生する。

【0344】PFNと論理アドレスのoffsetの間では、有効なビットが重なり合うことはないため、アラインメントの問題は発生しない。PTEのRL, T, ALの値と、実際にそれをアクセス可能なリングとの関係は、具体的には図312のようになる。

図において

・R0はring 0からのアクセス権、R1はring 1からのアクセス権を示す。ring 0～ring 3の区別はPSW中のRNGフィールドで示される。

・Rは読みだし可能、wは書き込み可能、Eは実行可能を示す。T=00, AL≠00の場合、ALが特別な意味を持ち、図313のような動作をする。この時、offsetの値は関係しない。このうち、T=00, AL=01の場合には、ハードウェアで使用しないPTBのフィールドとRLフィールドを、OSから自由に利用することができる。T=01を指定すると、読みだしはできないが実行は可能であるというページを作ることができる。これは、プログラムのコピーを禁止し、プログラムの実行に対する課金メカニズムを導入することを意図

したものである。一方、T=00, T=10を指定すると、読みだしや書き込みは可能であるが、実行は禁止であるというページを作ることができる。この機能を利用すれば、プログラムカウンタがデータ領域に飛び込んで来た場合に、それをチェックしてプログラムの暴走を阻止することができる。実行を禁止する機能は、メモリのデータ保護のための機能というよりも、デバッグのための機能と考えることができる。書き込みが可能である場合には、からなず読みだしも可能になっている。

【0345】A3-7. 64ビットへの拡張性

SR/URの切り換えビットを論理アドレスのMSBに固定すると、64bitへの拡張時に問題が生じる可能性がある。本発明装置では、論理アドレスを符号付きの数と考えることによって、この問題に対処する。SRとURの双方を32ビットから64ビットに拡大するためには、アドレス空間が二方向に伸びればよいわけである。そこで、アドレスを符号付きの数と考え、UR領域が正方向にSR領域が負方向に伸びると考えれば、この問題は解決される。具体的には、32→64の拡張に対

して、論理アドレスは符号拡張するようにしておく。メモリマップは図314のようになる。あるいは、図の書き換え方をかえて、図315のようになる。アドレスを符号付きと考えることにより、SR、UR双方の領域で拡張に対する連続性が保たれる。その代わり、H'80000000におけるOS領域とユーザ領域の接点が切られることになるが、これは問題ないと考えられる。なお、本発明装置の16ビット絶対アドレッシングモード(@ads:16)で、論理アドレスを符号拡張するようになっているのも、アドレスを符号付きとする考え方を適用したものである。

【0346】A3-8. MMU機能のバリエーションとLSIDの機能

LSIDの機能は〈〈L2〉〉であるため、最初のチップでは実装されず、将来のチップで導入される可能性が高い。したがって、最初に出たチップのためのLSIDの機能を利用しないプログラムも、将来のLSIDの機能の実装を見越したものにしておくのが望ましい。かといって、最初のチップでも不必要なオーバーヘッドは避けなければならない。そこで、こういったLSID(論理空間識別子)機能の有無やMMU機能の各種のバリエーションと、プログラム(OS)の互換性との関係について、検討を行なう必要がある。一般的な話として、MMU関係の仕様のバリエーションに対する対応には次の2つの方針がある。

①互換性を維持するため、実装していない機能であっても、縮退した機能でそのまま実行する。この場合、性能は落ちるだけで、オブジェクトレベルの互換性は達成される。

②実装していない機能については、EITで検出する。EIT起動の目的は、まちがいの検出とエミュレーションであるが、エミュレーションでは極端な性能の低下を招く場合があるので、実際問題として、オブジェクトレベルで変更の必要な場合が多い。

したがって、オブジェクトレベルの互換性は難しいが、あらかじめそれぞれのバリエーションの仕様が明らかになっていれば、ソースレベルで両方の仕様に対応できるようなプログラムを書くことは難しくない。例えば、PSTLB命令の機能を実装しない場合に、PSTLBをPTLBとして実行するのは①の方針であり、EIT(RIE9とするのは②の方針となる。①か②かは個別に検討する必要がある。

【0347】・PSTLBの/STオプションについて
TLBやキャッシュのインプリメント方式によっては、/STオプションの実装が難しい場合がある。しかし、/STオプションの指定は、TLBのページ範囲だけに影響するので、互換性維持のために①の方針をとる。すなわち、/STオプションの実装が難しい場合には、EITではなく/AT想到の動作を行なうということにする。

・LSID機能の実装とPTLB, PSTLBの/SSオプションについて

PTLB, PSTLBの/SSオプションの実装については、TLBやキャッシュのタグにLSIDの値を含めない場合にも、/SSオプションで全ページすることにより、①の方針に合わせることは可能である。しかし、LSID制御レジスタに値を書き込んだり、それを読みだしたりすることを考えると、オブジェクトレベルでの完全な互換性を達成するためには、LSID制御レジスタに相当するものを設けておく必要がある。ところが、互換性の確保だけのためにレジスタを一実装するのは無駄が多い。また、ハードウェアだけではなくソフトウェアに関しても、OSの中でLSIDの操作を行なう部分は、LSIDの機能を活かさなければ無駄な処理になってしまう。したがって、LSIDの有無については②の方針をとる。それに合わせて、/SSオプションの実装についても②の方針とする。

【0348】・LSID機能の実装とPTLB, PSTLBの/SSオプションについて

PTLB, PSTLBの/SSオプションの実装については、TLBやキャッシュのタグにLSIDの値を含めない場合にも、/SSオプションで全ページすることにより①の方針に合わせることは可能である。しかし、LSID制御レジスタに値を書き込んだり、それを読みだしたりすることを考えると、オブジェクトレベルでの完全な互換性を達成するためには、LSID制御レジスタに相当するものを設けておく必要がある。ところが、互換性の確保だけのためにレジスタに関しても、OSの中でLSIDの操作を行なう部分は、LSIDの機能を生かさなければ無駄な処理になってしまう。したがって、LSIDの有無については②の方針をとる。それに合わせて、/SSオプションの実装についても②の方針とする。具体的なLSID関連の仕様は、次のようになっている。

—LSIDの機能の有無と/SSオプションの実装の有無を一対一に対応させる。LSIDのあるプロセッサは/SSオプションが利用できるし、LSIDのないプロセッサは/SSオプションを利用できない。

—LSIDのあるプロセッサは、LSIDのないプロセッサに対してオブジェクトレベルで完全上位互換とする。つまり、LSIDのないプロセッサのために書かれたOSは、LSIDのあるプロセッサの上でもそのまま動く。ただし、LSIDの機能は生かすことができない。

—LSIDのあるプロセッサのために書かれ、LSIDの機能を生かすようなOSは、LSIDのないプロセッサでは動かない場合がある。具体的には、LSID制御レジスタの際に予約機能例外(RFE)が発生したり、

50 /SSオプションを指定した命令を実行した際に予約命

令例外 (RIE) が発生したりすることになる。

【0349】また、MMU関連の命令におけるLSID関係の仕様は、次のようになっている。

—LSIDの時は、LSIDのないときPSTLB/AS/AT @uraddr相当の動作を行なう。また、LSID実装時は、その機能を生かすため、LDCTXでTLB、キャッシュのページは行わない。LSID実装時は、TLBやキャッシュのタグ部分にLSIDまで含まれているので、LDCTXによりLSIDが変更されると、LDCTX実行前の論理空間でヒットしていたエントリがLDCTX実行後の新しい論理空間ではヒットしなくなる。ヒットしなくても、新しいエントリのロードに伴ってリプレースされない限りページはされない。再びLDCTXが実行されて以前の論理空間に戻った場合には、そのエントリがそのまま有効になる。

(LDCTXでキャッシュやTLBをページしていたのでは、LSIDの意味がなくなってしまう。)

—LDATE/PTでは、LSIDのない時PSTLB/AS/PT相当の動作を行なう。LSID実装時、prgaddrがSRであればPSTLB/AS/PT相当の動作を行ない、prgaddrがURであればPSTLB/SS/PT (LSID_of_TAG=R0) 相当の動作を行なう。prgaddrがSRかURかで動作が異なっているように見えるが、これはLDATEの問題ではなく、PSTLBの動作がURとSRで異なっているためである。LDATE命令自体の動作は、ATEの変更に伴って影響を受けるTLBやキャッシュをページする、ということで一貫している。

—LDATE/STでは、LSIDのない時PSTLB/AS/PT相当の動作を行なう。LSID実装時、prgaddrがSRであればPSTLB/AS/PT相当の動作を行ない、prgaddrがURであればPSTLB/SS/ST (LSID_of_TAG=R0) 相当の動作を行なう。(PTと同様)

—LDC命令でUATB、SATBを変更した場合は、仮想的な命令LDATE/ATを実行したと考えれば良く、LDATE/PT、LDATE/STと同様の動作を行なう。すなわち、LSIDのない時、SATBを変更した場合にはPSTLB/AS/AT (prgaddr=SR) 相当の動作を行ない、UATBを変更した場合にはPSTLB/AS/AT (prgaddr=UR) 相当の動作を行なう。LSID実装時、SATVを変更した場合にはPSTLB/AS/AT (prgaddr=SR) 相当の動作を行ない、UATBを変更した場合にはPSTLB/SS/AT (LSID_of_TAG=LSID, prgaddr=UR) 相当の動作を行なう。

【0350】付録4. 本発明装置のフラッグ変化
各命令のフラッグ変化の表記法は、以下の通りとする。

- 変化なし
- + 本来のフラッグの意味に合わせて変化する
- * 本来のフラッグの意味とは異なった変化をする
- 0 0にクリアされる

1 1にセットされる

【0351】A4-1. データ転送命令

図316に示す。

【0352】A4-2. 比較・テスト命令

10 図317に示す。

【0353】A4-3. 算術演算命令

図318に示す。ADDX, SUBXのX_flagは、destのサイズからの桁上げ、桁下げを示す。SUBXでsrcとdestのサイズが等しい場合、X_flagは符号なし演算としての大小関係といった意味にもなる。一方、L_flagの方は符号付き演算としての大小関係を表わす。MUL, MULU, MULX, DIV, DIVU, DIVX, REM, REMU, NEGのM_flag, Z_flagは、オーバーフローの発生にかかわらずdest設定値が基準である。MULX, DIVXのM_flag, Z_flagは、reg設定値には関係しない。DIVのV_flagは、0除算または(最小負数) ÷ (-1) の時にセットされる。DIVUのV_flagは0除算の時にセットされる。DIVXのV_flagは、0除算または商がdestのサイズに入らない時にセットされる。NEGのV_flagは、destが最小負数であった時にセットされる。INDEXのM_flag, Z_flagは、xreg設定値(結果の一部)が基準である。L_flagは結果が負であること、V_flagは、乗算または加算でのオーバーフローを示す。

【0354】A4-4. 論理演算命令

図319に示す。NOTのM_flag, Z_flagは、dest設定値(反転結果)が基準である。

【0355】A4-5. シフト命令

図320に示す。M_flag, Z_flagはdest設定値(シフト結果)が基準である。X_flagは最後にシフトアウトされた値が入る。SHA, SHL, ROTでcountが0の場合には、X_flag=0となる。SHAでは、count>0で符号の変化があった場合にのみV_flag=1。それ以外の場合はV_flag=0。

【0356】A4-6. ビット操作命令

図321に示す。

【0357】A4-7. 固定長ビットフィールド命令

図322に示す。固定長ビットフィールド命令では、BFCMP, BFCMPUがCMP, CMPUに準じたフラッグ変化をし、それ以外の命令がMOV, MOVUに準じたフラッグ変化をする。BFINS, BFINSUの場合、図323のBBBBBBBBがフラッグ変化の

277

基準となる。また、BFEXT、BFEXTUでは、抽出されたビットフィールドではなく、デスティネーションに設定される値を基準としてフラッグ変化する。これは、MOV命令などにおいて、デスティネーション側に設定された値を基準としてフラッグ変化するのに合わせたものである。

【0358】A4-8. 任意長ビットフィールド命令
図324に示す。

【0359】A4-9. 10進演算命令
図325に示す。BCD数は符号拡張が無意味なので、基本的に符号なしの数扱うものと考え、ADDU、SUBUに準じたフラッグ変化とする。なお、ADDX、SUBXは、符号なし、符号付きの両方の数扱うため、多少変則的なフラッグ変化になっており、ADDU、ADDDX、SUBU、SUBDXとは異なったフラッグ変化である。本発明装置では10進演算はサポートしない。

【0360】A4-10. スtring命令
図326に示す。SMOV、SCMP、SSCHのF__flagは、終了条件による終了（SSCHの場合はサーチ成功）を示す。V__flagは、エレメント数によって命令を終了した場合を示す。M__flagは、複数の終了条件を区別するために用いる。R3に関係する条件で終了した場合には0、それ以外の0またはR4の関係で終了した場合（〈〈L2〉〉のみ）には1となる。SCMPのX__flag、L__flag、Z__flagは、最終エレメントの比較結果を基準としてセットされる。X__flagはエレメントを符号なしデータと考えた時の大小、L__flagはエレメントを符号付きデータと考えた時の大小を示す。

【0361】A4-11. キュー操作命令
図327に示す。QINSのZ__flagは、空のキューに挿入したことを示す。QDELのZ__flagは、エントリの削除の結果キューが空になったことを示す。また、QDELのV__flagは、空のキューからエントリを削除しようとしたことを示す。QSCHのF__flagは、終了条件による終了（サーチ成功）を示す。V__flagは、キュー終了値R2による終了（サーチ失敗）を示す。M__flagは、複数の終了条件を区別するために用いる。R3に関係する条件で終了した場合には0、それ以外の0またはR4の関係で終了した場合（〈〈L2〉〉のみ）には1となる。

【0362】A4-12. ジャンプ命令
図328に示す。ジャンプ関係の命令では、フラッグは全く変化しない。

【0363】A4-13. マルチプロセッサ命令
図329に示す。

【0364】A4-14. 制御空間、物理空間操作命令
図330に示す。LDCでdestにPSWを指定した場合には、全フラッグが変化する。

278

【0365】A4-15. OS関連命令

図331に示す。本発明装置ではJRNG、RRNGはサポートしない。

【0366】A4-16. MMU関連命令

図332に示す。ACS命令のM__flagはread可、L__flagはexecute可、Z__flagはwrite可を示す。MOVPAのV__flagは、ページフォールトまたはエラーにより物理アドレスが得られなかったことを示す。F__flagは、ページフォールトを示す。LDATE、STATEのV__flagは、ページフォールトまたはエラーによりATEの転送ができなかったことを示す。本発明装置ではACS命令以外のMMU関連命令をサポートしない。

【0367】付録5. 異種サイズ間の演算について
本発明装置では、異なるバイト数の整数の間で各種の演算を行なうことができ、これを「異種サイズ間の演算」と呼んでいる。「異なるサイズ」といっても、現在は整数のみを対象としているため、データサイズの変換はゼロ拡張または符号拡張のみの簡単な処理で済む。例えば、32ビットの整数に8ビットの符号付き整数を加える場合、8ビット整数の符号ビット（MSB）を上位ビットにまで拡張してから加算を行なうわけである。符号拡張処理はゲート1～2段で可能なので、一般の加算命令と比べてそれほど複雑なわけではない。

【0368】A5-1. 異種サイズ間の演算の有用性
異種サイズ間の演算は、次のような場合によく使用する。

①オペランドの一方がイミディエート値の時
変数と定数の演算を行なう場合、定数のサイズはコンパイル時にわかるため、定数の方を小さいサイズとして扱えば、命令長を減らすのに効果がある。例えば、32ビットレジスタに8ビット定数の100を加える場合、32ビットの加算命令を使うと定数100を指定するのにも32ビットのフィールドが必要である。しかし、32ビットに8ビットを加算する命令を使えば、定数100を指定するフィールドが8ビットで済むため、命令が短くなる。さらに、乗除算の場合には、命令長だけではなく性能面でも影響がある。マイクロプロセッサでは32～64ビットの並列乗算器を持つのは苦しいので、どうしても加算とシフトで乗算を行なうことになるが、乗算の計算量は、2つのオペランドサイズの積に比例して多くなるため、2つのオペランドのうち的一方だけでもサイズの小さい方が有利である。異種サイズ間演算の機能がない場合には、例えば、32ビットの変数に3を掛けただけでも32ビット*32ビットの乗算を行なわなければならない。

②アドレス計算

アドレスの計算では、演算のデスティネーションのサイズをアドレス幅と同じにする必要がある。したがって、32ビットプロセッサの場合、32ビットと他のサイズ

279

との演算をよく行なう。例えば、文字の変換テーブルなどにおいて、テーブルのインデックスの範囲が8ビットにおさまる場合、インデックスとベースアドレスとの加算は、8ビット符号なし整数と32ビット整数との加算として実現される。

③高級言語

一般に、高級言語では、サブルーチンパラメータのサイズを必ずマシンの基本サイズ（例えば32ビット）に拡張することが多い。これは、スタックを使ってサブルーチンパラメータの受け渡しをするため、また分割コンパイル等を容易にするためである。さらに、言語Cのように、式の中の変数のデータサイズにかかわらず、式の評価は必ずマシンの基本サイズで行なうという場合もある。一方、メモリ上の変数、特に配列では、メモリ領域の節約のため、必要最小限のサイズとするのが普通である。したがって、配列とサブルーチンを同時に使用するプログラムでは、データの移動中または演算処理の途中のどこかでサイズの変換を行なわなければならない場合＊

B: Byte 8ビット

H: Halfword 16ビット

W: Word 32ビット

MOV src.B, dest.W
8ビットのsrcを32ビットに符号拡張してdestに転送。

MOV src.W, dest.B
srcの下位8ビットをdestに転送。
32ビット符号付き整数としてのsrcの値と、8ビット符号付き整数としてのdestの値が異なる時は、オーバーフローとなる。

ADD src.B, dest.W
8ビットのsrcを32ビットに符号拡張してdestに加算。

ADD src.W, dest.B
destにセットされる値は、srcの下位8ビットをdestに加算したものと同じである。しかし、命令の意味としては、src（32ビット）とdest（8ビットを32ビットに符号拡張）を加算し、それを8ビット符号付き整数に変換してdestに格納するということである。したがって、32ビットの和がdestの8ビットで表現できない値になった時は、オーバーフローとなる。

本発明装置では、ソースとデスティネーションのデータサイズが異なる場合に、通常符号拡張を行なう。ただし、特にゼロ拡張も必要と考えられる命令（MOV, CMP, ADD, SUB）については、ゼロ拡張と符号拡張を命令レベルで切り分け、MOVU, CMPU, AD

280

＊が出てくる。式の評価とサイズの変換を同時に行なうためには、本発明装置のような異種サイズ間の演算が便利である。

【0370】A5-2. 本発明装置における実際

本発明装置では、異種サイズ間の演算をサポートするため、データサイズの指定に関する直交性が非常に強くなっており、2オペランド一般形（G-format）の基本演算命令のほとんどで異種サイズ間の演算が可能である。つまり、2オペランド一般形の基本演算命令では、ソースのサイズとデスティネーションのサイズが独立に指定でき、必要に応じて符号拡張、ゼロ拡張、上位ビットの切り捨て、などを行なうようになっている。デスティネーションのサイズがソースのサイズより小さい場合にも演算は実行され、デスティネーションのサイズに従ってオーバーフローが検出される。

【0371】以下に、各命令における異種サイズ間演算の実際例を述べる。

DU, SUBU命令としている。MOVU, CMPU, ADDU, SUBU（加えてMULU, DIVU）では、デスティネーションのサイズがソースのサイズより大きい時にゼロ拡張を行ない、結果を符号なし整数と考えてオーバーフローの検出をする。

281

【0372】A5-3. 異種サイズ間の論理演算
 論理演算の場合は各ビットが全く独立なので、異種サイズ間の演算は意味がないし（フラグの変化を除けば、小さい方のサイズで行なうのと同様）、論理演算のオペランドに対するゼロ拡張や符号拡張もほとんど意味を持たない。しかし、例えばCで次のような関数を書いた場合には、意味はなくても
 符号拡張→論理演算
 というオペレーションを実行しなければならないことがある。

```

foo( ) {
    short    int16;
              /* 16 ビット符号付き整数 */

    int      int32;
              /* 32 ビット符号付き整数 */

    int32 &= int16;
              /* int16は符号拡張される */

}

```

ただ、このような例は、言語としての規則性や対称性のためにそう決まっているだけであり、一部のトリッキーなプログラムを除けばほとんど使わない機能であると言える。

【0373】論理演算の異種サイズ間演算をサポートするかどうかについて、問題点をまとめると以下のようになる。

①実行時

異種サイズ間の論理演算は、頻度としては非常に少なく、論理的な意味も持たない。本質的に他の命令で代用できるか、あるいはトリッキーなプログラムでしか使わない。

②コンパイル時

C言語では、論理演算でもゼロ拡張や符号拡張が必要になることがある。あまり使わない機能であっても、コンパイラとしては必ず正しいコードを出す必要がある。命令の対称性が重要。

③チップのインプリメント

符号拡張／ゼロ拡張の区別が全命令で統一的行なわれていれば、インプリメントの規則性の面から、論理演算でもゼロ拡張や符号拡張を導入するメリットがある。しかし、そのためには命令の割り当てに多くのビットパターンが必要となり、命令のエンコーディングが苦しくなってしまう。現実的には、論理演算を含む全命令で符号拡張／ゼロ拡張の区別を行なうことはできず、論理演算の符号拡張やゼロ拡張に対してインプリメントの規則性の面からのメリットは得られない。また、この部分はメーカー間で見解の異なる可能性もあるため、仕様を合わ

282

せるのが難しい。

【0374】結局、以上の中で②と③のどちらを重視するかということになるが、実質的な性能向上をねらうのであれば、やはり③を選択するのが適当と考えている。つまり、

・異種サイズ間の論理演算のように、実行する意味の少ない演算が足を引っ張ることによって、性能向上がはばまれるのはよくない。

・②の問題に関しては、符号拡張を含む異種サイズ間の論理演算は頻度の少ない演算であるから、多少実行速度が落ちてよい。

例えば、

```

AND    src.B, dest.W

```

srcを符号拡張

の代わりに

```

MOV    src.B, @-SP.W

```

src を符号拡張

```

AND    @SP+.W, dest.W

```

20

とすれば、多少実行速度は落ちるが、符号拡張～演算命令の統一的な置き換えが可能である。こうすれば、コンパイラの負担はそれほど増えない。したがって、本発明装置の仕様としては、異種サイズ間の論理演算をサポートしていない。異種サイズ間の論理演算に相当する命令ビットパターンを実行した場合には、動作を保証しないということになっている。

【0375】A5-4 異種サイズ間演算機能のまとめ
 本発明装置でサポートする命令と、整数のデータタイプとの関係をまとめると、次のようになる。

30

・8, 16, 32, 64ビット長の整数をサポートする。

・符号付きの整数を優先してサポートする。

・符号付き整数の算術演算に関しては、2オペランド命令において異種サイズ間の演算がサポートされている。ソースのサイズとディスティネーションのサイズは完全に独立に指定でき、サイズの制限はない。ソースの方がサイズが小さい場合は、符号拡張される。結果は符号付き整数として扱われ、それによってフラグ類がセットされる。

40

・符号なし整数の演算は、一部の命令（MOV, CMP, ADD, SUB, MUL, DIV）でのみサポートされている。サイズに関してはやはりソースのサイズとディスティネーションのサイズが完全に独立に指定できる。ソースの方がサイズが小さい場合は、ゼロ拡張される。結果は符号なし整数として扱われ、それによってフラグ類がセットされる。

50

・符号付きの整数と符号なしの整数の混在した演算はできない。ただし、加算命令などの場合は、ディスティネーションの符号の有無はフラグに影響するだけなので、

フラッグを見ないのであればADDまたはADDUで代用できる。

・異種サイズ間の論理演算はサポートしない。

【0376】付録6. 高級言語向きサブルーチンコール
高級言語におけるサブルーチンコールでは、単なるリターンアドレスの退避だけではなく、フレームポインタの設定、ローカル変数の領域の確保、汎用レジスタの退避といった処理も必要である。これらの処理はJSR, STMなどの命令に分解することも可能であるが、頻度が多いこと、処理が定型的であることにより、一つの命令(ENTER, EXITD)としてまとめられている。

【0377】A6-1. 本発明装置でのサブルーチンコール

高級言語(特にC, PASCAL)のサブルーチンコールでは、普通図333のような処理を行なう。本発明装置では、高級言語用サブルーチン命令ENTERとリターン命令EXITDを設けている。いくつか注意する点を述べる。

・FP(フレームポインタ)とディスプレイレジスタ
PASCALのようにスタティックスコープがある言語では、中間レベル(ローカル変数とグローバル変数の中間のレベル)の変数のアクセスにディスプレイレジスタを用いる。本発明装置のようにレジスタの多いプロセッサでは、汎用レジスタ上にこのディスプレイレジスタを置くのが有効である。これは、複数のFPを持つことに相当する。(実現方法は後述)

【0378】・パラメータ

パラメータを渡すには、バケットにしてその先頭アドレスをレジスタ等で渡す方法、パラメータをスタックに積む方法、等があり、高級言語では後者の方が一般的である。呼ばれたサブルーチンの側でスタック上のパラメータをアクセスするにはFP相対のモードを使うことが多い。サブルーチン実行後には、呼んだ側でスタック上のパラメータを解放する必要がある。言語によって、また分割コンパイルをしなれば、呼ばれた側で正確なパラメータ数がわかっているのも、リターン命令の中で解放するパラメータ数(SPへの加算値)を指定することができる。本発明装置では、この目的でEXITD命令を設けている。パラメータ数が動的に決まる場合もあるので(例えば、パラメータ数をサブルーチンに知らせるために、特定のレジスタやスタックを使う場合など)、EXITD命令におけるSPへの加算値としては、イミディエート値だけではなくレジスタ上の値を使用することもできる。しかし、言語Cのようにサブルーチンのパラメータの個数が確定しない言語では、サブルーチンを呼ばれた側からは、サブルーチンを呼ぶ側で決めたパラメータの個数がわからない。したがって、呼ばれた側で実行するEXITD命令では、解放するパラメータの個数を指定できない。その場合には、サブルーチンを呼んだ側でADD #n, SPを行なってパラメータを解放す

る。

【0379】結局、本発明装置のENTER命令では前の図の2.~4.の処理、EXITD命令では5.~7.の処理、または5~8の処理(ただし、8.で解放するパラメータ数はサブルーチン側で指定)を行なうことになる。1.はJSRと同じ処理になり、8.はサブルーチンを呼んだ側でADD ***, SPを行なう。本発明装置における高級言語でのスタックフレームは、図334のようになる。

・local variablesとparametersがなるべくFPから近い配慮となるように、ローカル変数確保よりレジスタセーブを後にした。

・EXITD命令にはPCのリストア(RTS)の操作まで含めている。

命令シーケンスの実際

(サブルーチン側でパラメータ数がわからない場合)図335に示す。

(サブルーチン側でパラメータ数がわかる場合)図336に示す。

【0380】A6-2. ブロック構造言語のためのディスプレイレジスタの構成例

ENTER-EXITDで処理するFPレジスタをダイナミックリンクとして利用するためには、内側のブロック(最高のレキシカルレベル)に対するフレームポインタにFPレジスタを割り当てるのがよい。その他のレキシカルレベルのフレームポインタには、値の変化の少ない順にR13, R12, R11...を使う。つまり、レキシカルレベルの高い内側のブロックへ行くにしたがって番号の小さいレジスタを使い、最小番号のレジスタの内容をFPと同じにする。各サブルーチンでは、ENTER命令実行後、FPを自分のレキシカルレベルに対応したフレームポインタ用レジスタにコピーし、その番号以上のレジスタをディスプレイレジスタとして、その番号以下のレジスタを評価用として利用すればよい。ただし、新しく書き換えたレジスタは必ず退避して値を保存しなければならない。

【0381】プログラム例(スタティックスコープ)

図337に示す。

実行状態の例(ダイナミックリンクとディスプレイレジスタ)

図338に示す。

- proc0*.var0*

proc0であるが、再帰呼び出しのため、最初のproc0とは異なったフレームになっている。

・FPのコピーにより破壊するレジスタは、コピー前にすべてENTER命令で退避しなければならない。レジスタを退避しておけば、サブルーチンの実行が終って一つ前の関数に戻った時、レキシカルレベルの大小にかかわらずディスプレイはもとの値に戻る。上の例では、レジスタの使い方について次のような関係が成立する。レ

285

キシカルレベルnのサブルーチン実行に関して

①R13...R13-n+1のn個のレジスタはディスプレイレジスタとして参照のみを行ない、書き込みはしない。

②R13-nのレジスタは、このレベルのローカル変数のディスプレイとして使用するため、ENTER実行後FPからコピーする。このディスプレイは、このサブルーチン実行中にさらに高いレベルのサブルーチンを呼んだ時に、呼ばれたサブルーチンからこのレベルの変数をアクセスするために使用する。このサブルーチンからこのレベルの変数をアクセスするには、同じ内容であるFPを使うのが良い。

③R13-n-1...R0の(13-n)個のレジスタは、レジスタ変数や評価用として利用する。

④R13-nのレジスタとR13-n-1...R0のうちで使用するレジスタは、必ずENTER命令で退避する必要がある。全部のレジスタが保存されなければならない。

【0382】付録7. 制御レジスタと制御空間

制御レジスタ関係の仕様は、チップバス(コプロセッサやキャッシュ、TLBなどに接続されるバス)やインプリメント方法との関係が深いため、〈〈LA〉〉仕様となっている。

【0383】A7-1. 制御空間の考え方

「本発明装置」では、チップバス上のメインプロセッサやコプロセッサに含まれるすべてのレジスタ、MMUやキャッシュ、TLBなどの制御レジスタ、およびチップバス上に接続されたコンテキストスイッチ用高速メモリに一意的なアドレスを付け、これを制御空間と呼ぶ。

「本発明装置」の制御空間は、従来のプロセッサに見られたコプロセッサ用のアドレス空間(Coprocessor-IDなど)をメインプロセッサの制御レジスタのアドレスと統合化、一般化したものであり、次のような特徴を持つ。

・「本発明装置」の制御空間には次のようなものが含まれる。

メインプロセッサの制御レジスタ

PSW, 各リングのスタックポインタなど

MMU関係の制御レジスタ(「本発明装置」はMMUを内蔵しない)

UATB, SATBなど

インプリメントに依存して必要となるレジスタ

[コプロセッサの制御レジスタ]

[コンテキスト退避用の高速メモリ] 将来チップに内蔵することを狙ったもの

[プロセッサ内の汎用レジスタ、テンポラリレジスタ]

外部からの診断、デバッグの目的

・制御空間は、コンテキスト(プロセスやタスク)間で共通の空間である。制御空間のアクセスはアドレス変換を伴わないため、単純化されたプロトコルで高速のアク

286

セスが可能である。この機能は、特に高速コンテキストスイッチでも利用される。制御空間の考え方は、将来、コプロセッサやコンテキスト退避用メモリがメインプロセッサに内蔵された時に、特に有効になると考えられる。しかし、最初のバージョンのチップでは、インプリメントの制約上、あるいはチップバスの構成上、制御空間の操作を統一的行なうのが難しい場合があるので、将来に備えてアドレスの割り当てのみを決めておき、制御空間操作命令はいくつかの制限を付けて使用してもよいということにする。

【0384】具体的には、次のような制限が付く。

・プロセッサの診断用の目的で、R0~R15やPCにも制御空間のアドレスが割り当てられているが、これは〈〈L2〉〉であり、「本発明装置」では実装しない。

・LDC, STCは、本来メインプロセッサ内の制御レジスタ、FPUの制御レジスタ、コンテキスト退避用メモリなどを統一的にアクセスすることを狙ったものである。しかし、「本発明装置」ではインプリメントの制限により、メインプロセッサ内の制御レジスタ以外(実効アドレスH'0~H'07fff以外)のものについては、LDC, STCでアクセスできない。

・「本発明装置」の制御空間のアドレスでは、バイトアクセス、ハーフワードアクセスが利用できない。また、ワードアクセスにおいてアラインメントが強制される。

・コンテキスト退避用メモリは、制御レジスタを置く領域(H'0~)と重ねることはできない。コンテキスト退避用メモリとしては、H'ffff8000~H'ffffffffffのアドレスが割り当てられている(さらに拡張領域としてH'80000000~)ので、H'80000000~H'ffffffffff以外の値をCTXBBに設定してLDCTX/CS, STCTX/CSを実行した場合には、エラーとする。また、LDCTX/CS, STCTX/CSの機能自体も〈〈L2〉〉となっている。

【0385】・「本発明装置」ではLDCTX/CS, STCTX/CSをサポートしない。

—— : 必須の仕様〈〈L1〉〉

..... : アドレス割り当てのみ〈〈L2〉〉

図339に示す制御空間では、バイトアクセス、ハーフワードアクセスができないのにもかかわらず、バイトアドレッシングとなっている。これは、一般の命令で利用される論理空間と同じように、制御空間でも汎用アドレッシングによる実行アドレス指定が可能であり、論理空間と同じ形のバイトアドレッシングにしておかないと、混乱を招くからである。また、制御空間で汎用アドレッシングを利用可能としたのは、制御空間をコンテキスト退避に利用することを考えたためである。LDC, STCでメインプロセッサ内の制御レジスタしかアクセスできない場合には、バイトアドレッシングとした意味がな

287

くなってしまう、多少不自然な仕様になる。しかし、その背景には上記のような将来的な見通しがあり、一部の機能のみを実現した場合に多少不自然に見えるのはやむを得ない。

【0386】A7-2. メインプロセッサの制御レジスタ *

H' 0000~H' 03ff メインプロセッサ、

H' 0400~H' 07ff メインプロセッサ、

H' 0800~H' 0bff FPU(TRON reserve)

H' 0c00~H' 0fff FPU<LV>>

...

* はコンテキスト毎に別々に用意されるレジスタ

/ は必ずしも実装（アドレス割り当て）をしなくても良いレジスタ

288

*タ

制御レジスタのニモニクとアドレスについては、次のようになる。制御レジスタのアドレスが $8n+4$ の位置にあるのは、レジスタを64ビットに拡張することを考慮したためである。

MMU(TRON reserve)

MMU <<LV>>

【0387】

アドレス

レジスタ

H' 0000

reserved

H' 0004

*

PSW

H' 0008

reserved

H' 000c

、(*)

SMRNG

H' 0010

reserved

H' 0014

(*)

IMASK

H' 0018

reserved

H' 001c

reserved

H' 0020

reserved -- EITVBH

H' 0024

BITVB

H' 0028

reserved -- JRNGVBH

20

30

40

H' 002c

「本発明装置」

reserved -- JRNGVB

H' 0030

reserved -- CTXBBH

H' 0034

*

CTXBB

H' 0038

reserved

H' 003c

reserved

H' 0040

reserved -- SATBH

H' 0044

「本発明装置」

reserved -- SATB

H' 0048

reserved -- UATBH

H' 004c

*

「本発明装置」

reserved -- UATB

H' 0050

reserved

H' 0054

*

「本発明装置」

reserved -- LSID

H' 0058

reserved

H' 005c

reserved

H' 0060

reserved -- IOADDRH

H' 0064

/

IOADDR

H' 0068

reserved -- IOMASKH

H' 006c

/

IOMASK

H' 0060~H' 007f

、

reserved

【0388】

H' 0080

reserved

H' 0084

(*)「本発明装置」

reserved -- DCE

H' 0088

reserved

H' 008c

DI

H' 0090

reserved

	289	
H' 0094	* 「本発明装置」	
	reserved -- CSW	
H' 0098	reserved	
H' 009c	(*) 「本発明装置」	
	reserved -- CTXBFM	
H' 00a0~H' 00ff	reserved	
【0389】		
H' 0100	reserved -- SPIH	
H' 0104	SPI	
H' 0108~H' 011f	reserved	
H' 0120	reserved -- SPOH	
H' 0124	* SPO	
H' 0128	reserved -- SPIH	
H' 012c	* 「本発明装置」	
	reserved -- SPI	
H' 0130	reserved -- SP2H	
H' 0134	* 「本発明装置」	
	reserved -- SP2	
H' 0138	reserved -- SP3H	
H' 013c	* SP3	
H' 0140~H' 017f	reserved	
【0390】		
H' 0180	reserved -- ROH	
H' 0184	* 「本発明装置」	
	reserved -- RO	
H' 0188	reserved -- R1H	
H' 018c	* 「本発明装置」	
	reserved -- R1	
...	...	
H' 01e0	reserved -- R12H	
H' 01e4	* 「本発明装置」	
	reserved -- R12	
H' 01e8	reserved -- R13H	
H' 01ec	* 「本発明装置」	
	reserved -- R13	
H' 01f0	reserved -- R14H	
H' 01f4	* 「本発明装置」	
	reserved -- R14	
H' 01f8	reserved -- PCH	
H' 01fc	* 「本発明装置」	
	reserved -- PC	

10

20

30

40

50

290

【0391】	
H' 0200~H' 03ff	reserved
(H' 0400 ~H' 07ff	<<LV>>>
H' 0410	BBC
H' 0414	BBP
H' 0500	DBC
H' 0520	XBP0
H' 0524	XBP1
H' 0540	OBP0
H' 0544	OBP1

【0392】 A7-3. 制御レジスタの未使用ビット
制御レジスタのうち、使用していないビットについては、1を書き込んだ場合にそれをチェックしてEITとするのが望ましい。しかし、中途半端なチェックを行なうと、互換性（特に下位チップとの互換性）を保つのが難しくなること、チェックのためにオーバーヘッドが生じること、により、PSWを除いて未使用ビットのチェックは行なわないことにする。CTXBFMなど〈〈L2〉〉の機能を持つレジスタについても、〈〈L2〉〉が実装されていない場合にはエラーチェックが行なわれないし、書き込んだ値がそのまま読み出されるとは限らない。ただし、チェックが行なわれていなくても、空きビットには必ず'0'を入れてもらうように、マニュアル等で指導する必要がある。PSWについては、未使用のビット'-'に'1'を書き込もうとした場合に、予約機能例外(RFE)とする。

【0393】 以下の制御レジスタの内容の説明における'-'、'='、'*'のビットの意味は次のとおりである。

- 291
 ' - ' 0にreserved (違反時例外発生)
 ' + ' 1にreserved (違反時例外発生)
 このビットに0(1)を書き込むことは可能であるが、このビットに1(0)を書き込もうとするとLDC, LDCTX など予約機能例外(RFE)を発生する。
- 292
 ' = ' 0にreserved (違反時も無視)
 ' # ' 1にreserved (違反時も無視)
 このビットは0(1)を書き込むべきビットであるが、このビットに1(0)を書き込もうとしても単に無視され、このビットに0(1)を書き込んでも1(0)を書き込んでも動作は同じである。
 ユーザへのマニュアルには、将来の拡張のためこのビットに0(1)を書き込むように明記する。
- * 書き込み時の値は自由。
 ハードウェアの動作は' = ', ' # ' と同じであり、どのような値が書き込まれても単に無視する。ただし、このビットは' = ', ' # 'とは異なり、将来チップの機能が拡張された場合にも使用しないビットである。したがって、ユーザが自由な値を書き込んでも構わない。ユーザへのマニュアルの中でも、このビットは無視されるということを明記し、ビットのマスク処理などは省い

てもらうようにする。

・ IMASK, SMRNG, DI, DCE, CTXBF Mでは、未使用のビットが' * ' になっている。PSWでは、未使用のビットが' - ' になっている。それ以外の制御レジスタでは、未使用のビットは ' = ' である。

・ PSB, PSMの未使用フィールドも' - ' である。 30
 したがって、LDPSB, LDPSMでも予約機能例外(RFE)が発生する。

・ ' - ' のビットの読みだした場合には、' 0 ' が読み出される。' = ', ' * ' のビットを読みだした場合に得られる値は不定である。前に書き込んだ値がそのまま読み出されるときも限らない。

【0394】A7-4. 制御レジスタの内容
 PSW

図340に示す

Processor Status Word 40
 内容については本文を参照。

PSM, PSB

PSWのうち、ユーザのアクセス可能な下位の2バイトのみを抜き出したもの。LDPSB, LDPSM, STPSB, STPSM命令によってアクセスする。制御レジスタのうち、PSB, PSMのみがring 0以外からアクセスできる。

IMASK

図341に示す

PSWのうち、個別にアクセスすることの多いIMASKのフィールドを抜き出して別レジスタとしたものである。IMASKの操作を容易にすることと、性能向上を狙ったものである。IMASK以外のフィールドには、何を書き込んでも単に無視される。

SMRNG

図342に示す

PSWのうち、個別にアクセスすることの多いSMRNGのフィールドを抜き出して別レジスタとしたものである。SM, RNGの操作を容易にすることと、性能向上を狙ったものである。SMRNG以外のフィールドには、何を書き込んでも単に無視される。

【0395】CTXBB

図343に示す

Context Block Base

CTXBのベースアドレスを指すレジスタ。LDCTX, STCTX命令で使用される。CTXBBは「本発明装置」64への拡張を考え、「本発明装置」32でも8バイトのアラインメントを強制している。したがって、CTXBBの下位3ビットは' === ' となる。つまり、0にreservedであるが、違反時にも無視される。

DI (Delayed Interrupt)

図344に示す

293

294

DI(Delayed Interrupt)要求を示すレジスタ。

DI=0000 優先度0の外部割り込み(NMI)処理後のDI要求

DI=0001 優先度1の外部割り込み処理後のDI要求

DI=0010 優先度2の外部割り込み処理後のDI要求

...

DI=1110 優先度15の外部割り込み処理後のDI要求

DI=1111 DI要求なし

【0396】DI (Delayed Interrupt) は、ソフトウェアによって外部割り込みを発生させるメカニズムであり、非同期に発生する各種の処理要求をペンディングとしたい時や処理順序をシリアル化したい時に有効である。優先度の高い外部割り込みの処理が終わった後で別に起動したい処理がある場合、その要求をDIに登録しておくことによって自動的に処理が起動される。DIは、外部割り込みに対してDCEと同等の処理を行なうものである。REIT命令などによってPSWのIMASKが変化した場合に、DI<IMASKであればDIのEIT処理が起動される。このレジスタのDI以外のフィールドに何を書き込んでも単に無視される。

【0397】CSW

図345に示す

Context Status Word

このレジスタは、コンテキスト毎に切り換えの必要な情報のうち、ネストの行なわれないものを集めたものである。DCE (Delayed Context Exception) 要求を示すDCEフィールドと、CTXBのフォーマットを示すCTXBFMのフィールドから成る。CTXBFMの機能は付録8を参照のこと。CTXBFMの機能がインプリメントされない場合には、DCEレジスタとCSWレジスタが全く同じ情報を扱うことになるので、CSWレジスタもインプリメントされない(アクセス時RFE)場合がある。この時、CTXBに置かれるのは、形式的にはCSWレジスタであるが、実際はDCEレジスタとなる。CSWとDCE、CTXBFMの関係は、PSWとIMASK、SMRNGの関係に同じである。CTXBに置かれるのは、DCE、CTXBFMなどの情報を圧縮したCSWの方である。「本発明装置」ではDCE='111'に固定とする。

【0398】DCE

図346に示す

Delayed Context Exception
CSWのうち、個別にアクセスすることの多いDCEのフィールドを抜き出して別レジスタとしたものである。DCEの操作を容易にすることと、性能向上を狙ったものである。DCE以外のフィールドには、何を書き込んでも単に無視される。CSWレジスタがインプリメントされない場合、コンテキストスイッチ時にCSWレジ

スタの代わりにCTXBとの転送が行なわれるのは、このDCEレジスタである。この場合、コンテキスト退避の時は、'*'のビットがすべて0となってCTXBに書き込まれる。また、コンテキストをロードする時は、'*'の部分のビットの値はチェックされない。

【0399】CTXBFM

図347に示す

Context Block Format

CSWのうち、個別にアクセスすることの多いCTXBFMのフィールドを抜き出して別レジスタとしたものである。CTXBFMの操作を容易にすることと、性能向上を狙ったものである。CTXBFM以外のフィールドには、何を書き込んでも単に無視される。このレジスタは<L2>である。

EITVB

図348に示す

EIT Vector Base

EIT (例外、割り込み) ベクトルテーブルの先頭の物理アドレスを示す。EITVBは「本発明装置」64への拡張を考え、「本発明装置」32でも8バイトのアラインメントを強制している。したがって、EITVBの下位3ビットは'==='となる。つまり、0にreservedであるが、違反時にも無視される。

【0400】JRNGVB

図349に示す

JRNG Vector Base

JRNG命令のベクトルテーブルの先頭の論理アドレスを示す。JRNGVBによるテーブルのベースアドレスは、「本発明装置」64への拡張を考え、「本発明装置」32でも8バイトのアラインメントを強制している。また、JRNGVBのLSBはEnableビットとなっており、Eが0の時にはJRNG実行禁止となる。したがって、JRNGVBの下位3ビットは'==E'で表記されている。'='のビットは、0にreservedであるが、違反時にも無視される。

SP0~SP3

図350に示す

Stack Pointer for ring3

ring0~ring3で使用するスタックポインタ。

SPI, SP0~SP3についてはアラインメントの制約はなく、LSBまで有効である。

295

SPI

図351に示す

Stack Pointer for Interrupt

外部割り込み用のスタックポインタ。

【0401】IOADDR, IOMASK

図352に示す

アドレス変換を行なわない場合 (PSWのAT=00, 10) において、I/O領域の物理アドレスを指定するレジスタである。通常MMUでアドレス変換を行なう場合には、PTE中のNCビットによってI/O領域の指定を行なうが、システムスタート時などでアドレス変換ができない場合には、IOADDR, IOMASKの2つのレジスタを使ってI/O領域の指定を行なう。アドレス変換なしでメモリアクセスをする場合、物理アドレスとIOMASKの論理積がIOADDRと等しければ、そこはI/O領域と見なされる。その領域のデータについては、データのキャッシュへの取り込みやプリフェッチが行なわれず、命令の要求するメモリアクセスと実際の物理的なメモリアクセスが1対1に対応する。アドレス変換のある場合には、IOADDR, IOMASKレジスタは使用しない。また、プロセッサのインプリメントによってキャッシュやプリフェッチを行なわない場合には、必ずしもIOADDR, IOMASKレジスタを使用する必要はない。

【0402】UATB

図353に示す

Unshared region Address Translation Base

内容については付録3を参照。

SATB

図354に示す

Shared region Address Translation Base

内容については付録3を参照。

LSID (「本発明装置」では実装しない)

図355に示す

Logical Space ID

複数の論理空間の間の区別を行なう番号を入れる。複数の論理空間に属するTLBや論理キャッシュなどを混在させる時に、この番号を利用する。LSIDの有効なビット数については、インプリメント依存である。

【0403】付録8. 「本発明装置」のCTXB

A8-1. CTXBについて

「本発明装置」はMMUをもたないため「本発明装置」でサポートするCTXBフォーマットをどのようにするか現在検討中である。OSが、タスク、プロセスなどといった並行処理やルーチンの機能をサポートしている場合、PCや汎用レジスタなどのハードウェア資源の情報は、並行処理の単位となるそれぞれのプログラム毎に

296

別々に持つのが普通である。これらのハードウェア資源は、プロセッサと同様に時分割で使用されるため、現在実行中でないプログラムに対するハードウェア資源の情報は、メモリなどに退避しておく必要がある。「本発明装置」では、こういった並行処理の単位となるプログラムの流れをコンテキストと呼ぶ。また、それぞれのコンテキストを実行するために、ハードウェア資源の情報をメモリ上にまとめて退避したものをContext Block (CTXB) と呼ぶ。CTXB自体のおかれる空間は、LDCTX, STCTX命令のオプションとして、論理空間LS、制御空間CSより選択できる。OSの書きやすさという点ではLSを利用するのが適しているが、コンテキストスイッチを特に高速化したい場合や、コンテキストスイッチ退避用のメモリをチップ内に設ける場合には、CSを利用することもできる。ただし、CSの指定は、将来コンテキスト退避用のメモリをチップに内蔵した場合に有効に活用されるものであり、現在は〈L2〉となっている。また、「本発明装置」では、現在実行中のコンテキストのCTXBの先頭アドレスを保持するレジスタが設けられており、これをCTXBBase Register (CTXBB) と呼んでいる。「本発明装置」のCTXBのフォーマットは、次のようになっている。このうちの一部は、LDCTX, STCTX命令により、ハードウェアでサポートされている。

【0404】「本発明装置」32での標準CTXBフォーマット

図356に示す

一般に、コンテキストスイッチによって切り換える必要があるのは、OSのPCやPSWではなく、ユーザプログラムのPCやPSWである。ところが、ユーザプログラムのPCやPSWは、普通はOS呼び出し時にスタック中に退避されている。上記のCTXBフォーマットにおいて、PC, PSWがスタック中に置かれているのは、そのためである。SPIを使用した外部割り込みの処理ハンドラの最後で直接コンテキストスイッチを行なう場合は、上記のCTXBフォーマットを実現するために、PC, PSWを別命令で転送する必要がある。しかし、この場合はDCE, DIを活用し、外部割り込みから抜ける時にDCE, DIを使ってコンテキストスイッチを行なうという方法がある。そうすれば、DCEやDIでSPOを指定することにより、上記のデータ構造が自然に実現できる。

【0405】A8-2. CTXBの可変性

CTXBに含まれる情報のうち、' * 1 ' ~ ' * 5 ' の付いた部分は、システム構成などによって可変性のある部分である。これらの点について説明する。CTXBの内容やフォーマットは、以下のような要因によって動的に (あるいはコンテキスト毎に) 変化することがある。

297

OSの構成によっては、コンテキストスイッチでSP1～SP3の切り換えを行なっても意味のないケースが考えられるため、SPI～SP3を退避したくない場合がある。また、MMUを使用しない用途でのコンテキストスイッチでは、UATB、LSIDを切り換える必要がない。

(*1) JRNG～RRNGでは、外側のリングが内側のリングのスタックに退避されるため、現在リングよりも外側のリングのSPの値は意味を持たない。特に、ring0でのみ実行されるコンテキストスイッチの時点では、SP1～SP3の値が意味を持たない。SP1～SP3は、直接あるいは間接にSP0のスタックに保持されているため、SP0の切り換えによってSP1～SP3も間接的に切り換わることになるからである。一方、TRAPA～REITの中でコンテキストスイッチが起こった場合にはSP1～SP3の切り換えも必要である。したがって、SP1～SP3をCTXBに含めたい場合と、そうでない場合がある。

(*2) MMUを実装しない。〈〈L1R〉〉仕様では、UATBは不要である。

(*3) LSIDは、複数の論理空間を区別するための番号である。LSIDの実装は〈〈L2〉〉なので、LSIDがCTXBに含まれる場合と、そうでない場合がある。

【0406】・退避する汎用レジスタの指定(*4)

FR	FPU レジスタを退避することを指定
	「本発明装置」標準のFPU レジスタのコンテキスト退避を指定する。この機能は、特に、将来FPU がチップに内蔵された場合に利用する。
RG	R0-R14を退避することを指定
	この機能は、特に、将来コンテキスト退避用メモリがチップに内蔵された場合に利用する。
SP	SPの退避を指定
	SP=00 SP0, SP1, SP2, SP3を退避する
	SP=01 reserved
	SP=10 SP0, SP3を退避する(〈〈L1R〉〉用)
	SP=11 SP0のみ退避する
	この機能は、JRNGによりOSを呼び出す場合に、SP1～SP3の無駄な転送を行なわないために利用する。

298

コンテキストで使用しないレジスタやOSで使用するワーキングレジスタについて、CTXBとの退避、復帰を行なわなければ、無駄な転送がなくなり、コンテキストスイッチ時間が短縮される。

・コプロセッサ使用の有無(*5)

FPUのレジスタは汎用レジスタとは別になるが、これもコンテキスト情報として持つ必要がある。したがって、そのコンテキストがFPUを使用しているかどうかによって、CTXBが動的に変わるケースが生じる。

【0407】以上のようなCTXBのバリエーションに対処するため、「本発明装置」では、以下のような方法をとっている。

・最初のバージョンの〈〈L1〉〉のチップでは、LDCTX, STCTXでCSW, SP0～SP3, UATBの転送のみを行なう。R0～R14については、別命令LDM, STMを利用して転送し、(*4)に対処する。

・それ以外のCTXBのバリエーションに対しては、現在のCTXBのフォーマットを識別するレジスタ(CTXBFM)を導入し、このレジスタによって、CTXBに何が含まれ、LDCTX, STCTXで何を転送しなければならないかを知ることとする。なお、CTXBFMとDCEの情報を合わせたものは、CSWレジスタとしても扱われる。

【0408】[CTXBFM] 図357に示す

299

300

また、〈〈L1R〉〉でSP1, SP2を実装しない場合に利用する

MM MMU 関連レジスタの退避を指定
MM=00 UATB を退避
MM=01 UATB, LSIDを退避
MM=10 MMU関連レジスタは退避しない(〈〈L1R〉〉用)
MM=11 reserved

【ただし、CTXBFMの詳細は検討中である。】

【0409】〈〈L1〉〉で標準的なフォーマットのCTXBでは、LDCTX, STCTXにより、CSW (DCE, CTXBFM)、SP0~SP3, UATBが転送される。これは、CTXBFMをすべて0にすることによって指定される。LDCTX命令では、CTXBからフェッチした新しいコンテキストのCSWの中のCTXBFMを見て、CTXBの以下の部分のフォーマットを判断し、指定されたものをロードする。また、STCTX命令では、現在のCTXBFMの値を見て、指定されたものをCTXBに退避する。ただし、CTXBFMの機能は〈〈L2〉〉となっており、将来の拡張である。つまり、CTXB固定の仕様が〈〈L1〉〉、CTXB可変の仕様(上位コンパチ)が〈〈L2〉〉である。〈〈L1R〉〉のチップについては、SP1, SP2, UATBの転送が不要なので、CTXBにもこれらの値は含めない。CTXBにこれらのレジスタ値が含まれているかどうかの識別は、CTXBFMによって行なうことが可能である。ただし、CTXBFMの実装が重い場合には、LDCTX, STCTX命令の追加オプションにより直接CTXBのフォーマットを指定する仕様や、LDCTX, STCTX命令の追加オプションによりCTXBFMの有効性の有無を指定する仕様にすることも考えられる。

【0410】A8-3. ソフトウェアコンテキスト
プロセスやタスク毎に持つ情報の中には、OSがソフトウェアによって管理する情報も含まれている。これらの情報はOSによって一定していないので、当然ハードウェア(LTCTX, STCTX命令)ではサポートできない。これらの情報をソフトウェアコンテキストと呼ぶ。例えば、ITRONの場合、タスク状態、終了時処理ルーチンのアドレス、例外処理ハンドラのアドレス、wake upのカウント、キューの構成のためのリンク用領域などがソフトウェアコンテキストに含まれる。CTXBを論理空間(LS)に置いた場合には、汎用レジスタなどのハードウェアコンテキストとソフトウェアコンテキストを同じように扱うことができる。しかし、ハードウェアコンテキストとして、CSなどの別空間を使用した場合には、ソフトウェアコンテキストもCSに置くか(この場合は、LDC, STCなどの命令が有効である)、あるいは両者をポインタでつないでいて間接

10 参照するか、といった方法をとる必要がある。

【0411】付録9. 「本発明装置」のEIT処理
EIT処理の概要は以下のとおりであるが、細部の仕様は検討中である。通常のプログラム実行の流れをハードウェア機構によって中断し、それに割り込むような形で非同期に起動される処理を、「本発明装置」ではEIT処理と呼ぶ。EIT処理には、次のようなものが含まれる。

- ・内部割り込み(トラップ-trap)
- ・例外割り込み(例外-exception)
- 20 ・外部割り込み(割り込み-interrupt)

トラップ、例外、割り込みの区別は、プログラマから見たそのEITの発生原因によって行なわれるものであり、インプリメント上のメカニズムの違い(スタックに退避される情報の違いなど)を意味するものではない。プロセッサが命令実行中にEITを検出すると、シーケンシャルな命令の実行を中断してEIT処理を開始する。EIT処理では、EITを検出した時のプロセッサの状態をスタックに退避し、EITハンドラを起動する。ここまでのプロセッサのハードウェアによって行なわれる。一方、起動されたEIT処理ハンドラでは、EITに応じてエラーの回復、エラーメッセージの表示、エミュレーションなどの処理を行なう。EIT処理ハンドラは、ソフトウェアにより実現されるものである。大部分のEITでは、EIT処理ハンドラの最後でREIT命令を発行することにより、中断されたものの命令列に復帰し、処理を再開することが可能となっている。

「本発明装置」では、将来の機能拡張を考慮し、未定義の命令、不当な命令についてのエラー検出やエミュレーション用のメカニズムを強化する方針である。したがって、命令フォーマットの組み合わせにより不当なオペレーションになる場合や、インプリメントされていない機能を実行しようとした場合には、できるだけエラーとして検出し、例外割り込みを発生する。

【0412】A9-1. EITの種類

「本発明装置」で発生するEITには、次のようなものがある。

[メモリ、アドレス関係]

ページ不在例外(POE) 「本発明装置」では発生しない

50 Page Out Exception (POE)

301

命令またはオペランドアクセス時のアドレス変換において、UATB, SATB, STE, PTEのPIビットが0であった場合に発生する。意味的には、ページ不在、ページテーブル不在、セクションテーブル不在を含めたものである。いわゆるページフォールトの例外である。

アドレス変換例外 (ATRE)

Address Translation Exception (ATRE)

アドレス変換中のエラーによって発生する。具体的には、STE, PTEでreservedのビットパターンを使用していた場合、UATB, SATB, STE, PTEによって未使用領域となっている部分を参照した場合、リング保護に違反したメモリアccessを行なった場合などに発生する。EIT発生の原因や詳細な情報は、ATRE発生時にスタックに積まれる情報によって区別される。

【0413】バスアクセス例外 (BAE)

Bus Access Exception (BAE)
命令またはオペランドアクセスにおいて、一定時間以内にバスからの応答がなく、メモリアccessができなかった場合に発生する。いわゆるバスエラーである。

奇数アドレスジャンプ例外 (OAJE)

Odd Address Jump Exception (OAJE)

分岐命令で、分岐先のアドレスが奇数であった場合に発生する。この例外は、ジャンプ先をオペランドとして直接指定する命令 (JMP, ACB等)、スタックからリターンアドレスを得る命令 (RTS, EXITD, RRNG, REIT)、およびJRNG命令で発生し、EIT処理の起動時には発生しない。EIT処理起動時に新PCが奇数であった場合には、システムエラー例外 (SEE)となる。[JRNG, EITは詳細仕様調整中]

【0414】[命令、演算関係]

特権命令違反例外 (PIVE)

Privileged Instruction Violation Exception (PIVE)
ring 0以外から特権命令を実行しようとした場合に発生する。

〈〈L1〉〉機能例外 (L1E)

L1 function Exception (L1E)

〈〈L1〉〉機能のインプリメントされていないプロセッサにおいて、〈〈L1〉〉の機能を実行しようとした場合に発生する。〈〈L1〉〉を実装しているプロセッサであれば、この例外は発生せず、このEITに対するベクトル番号はreservedとなる。

予約命令例外 (RIE)

Reserved Instruction Exception (RIE)

302

現在割り当てられていない命令やアドレッシングモードのビットパターンを実行しようとした場合に発生する。いわゆる未定義命令の例外である。「本発明装置」32で64ビットのサイズを指定した場合、Pビットを1にした場合、未実装の〈〈L2〉〉命令を実行しようとした場合、未定義、未実装のオプションを指定した場合も、これに含まれる。また、命令によって禁止されているアドレッシングモードを使用した場合 (JMP命令におけるイミディエート指定など) や、インプリメントされていない段数の付加モードを使用した場合も、これに含まれる。

【0415】予約機能例外 (RFE)

Reserved Function Exception (RFE)

命令やアドレッシングモードのビットパターン以外で、将来の拡張のために予約されている機能を利用しようとした場合に発生する。例えば、PSWに関しては、XAやreserved (' - ') のビットに1を書き込んだ場合、SMRNGのフィールドに reserved の値 (SM, RNG=001など) を書き込んだ場合、非特権命令のLDPSB, LDPSMによってPSMやPSBのreserved (' - ') のビットに1を書き込んだ場合などに予約機能例外 (RFE) が発生する。このほか、実装されていない制御レジスタをアクセスしようとした場合や、WAIT命令でimask≥16を指定した場合にも、予約機能例外 (RFE) が発生する。なお、「本発明装置」では、命令ビットパターン (アドレッシングモードやサイズの指定を含む) のみでエラーと判定できるものを予約命令例外 (RIE) とし、アドレスやオペランド値によってエラーかどうかの状態が変化するものを予約機能例外 (RFE) としている。

【0416】コプロセッサ命令例外 (CIE)

Coprocessor Instruction Exception (CIE)

コプロセッサが接続されていないのに、コプロセッサ用に割り当てられた命令を実行しようとした場合に発生する。

コプロセッサコマンド例外 (CCE)

Coprocessor Command Exception (CCE)

コプロセッサとのインタフェースでエラーがあった場合に発生する。

コプロセッサ実行例外 (CEE)

Coprocessor Execution Exception

コプロセッサの命令実行においてエラーがあった場合に発生する。

【0417】不正オペランド例外 (IOE)

Illegal Operand Exception

303

(IOE)

不合理なオペランドの指定を行なった場合に発生する。固定長ビットフィールド命令で32(64)ビット以上のwidthを指定した場合などがこれに含まれる。奇数アドレスへのジャンプやゼロ除算も意味的には不正オペランド例外の一部と考えられるが、特別な意味を持つために異なる例外に分類されている。なお、「本発明装置」において命令のオペランドが不当な場合の対策としては、不当オペランド例外やゼロ除算例外とするケースのほかに、特にチェックを行なわないケース(CHK命令の上限値と下限値の大小関係など)、適当な解釈をしてそのまま命令を実行するケース(シフト命令でcountが大きい場合など)、などがある。一方、命令を実行した結果が不当(オーバーフローなど)な場合には、EITが起動されることはない。この場合は、Vflagをセットして命令を終了するケース(ADD, MOVなど多数)、特に何もしないケース(UNPKssでのオーバーフローなど)、などがある。

10進不正オペランド例外(DDE)

Decimal Illegal Operand Exception (DDE)

符号付き10進演算命令において、0~9以外のデータをオペランドとして指定した場合に発生する。この例外も、意味的には不正オペランド例外(IOE)の一部であるが、別の例外に分類されている。

【0418】予約スタックフォーマット例外(RSFE)

Reserved Stack Format Exception (RSFE)

REIT命令によってEITから復帰する際に、EITスタックフレームのフォーマットを示す番号(FORMAT)が、REIT命令で処理できないものであった場合に発生する。

リング遷移違反例外(RTVE)

Ring Transition Violation Exception (RTVE)

JRNG命令で外側のリングに移ろうとした場合、RRNG命令で内側のリングに移ろうとした場合など、不当なリング遷移を行なおうとした場合に発生する。なお、JRNG命令で参照すべきJRN RVTEを含むページが未使用領域となっていた場合には、リング遷移違反例外(RTVE)ではなくアドレス変換例外(ATRE)の未使用領域参照エラーが発生する。

ゼロ除算例外(ZDE)

Zero Divide Exception (ZDE)

0除算を行なった場合に発生する。

【0419】[デバッグ]

デバッグ例外(DBE)

Debug Exception (DBE)

304

デバッグに関係して発生する。具体的には、命令のシングルステップ実行やブレークポイントを実現するための例外であるが、詳細な仕様は〈LV〉である。

[トラップ]

無条件トラップ命令(TRAPA)

TRAPA Instruction

TRAPA命令により発生する。TRAPAのEITベクトルは、TRAPAのオペランドvectorに対応して16種類用意されている。

10 条件トラップ命令(TRAP)

Conditional TRAP Instruction

TRAP命令により発生する。

【0420】[DCE, DI]

遅延割り込み(DI)

Delayed Interrupt (DI)

DIレジスタ中のDIフィールドが、PSW中のIMASKフィールドよりも小さい値になった場合に発生する。このEITは、コンテキストとは独立した非同期の事象を処理するために有効である。DI処理のEITベクトルは、各割り込み優先度毎に15種類用意されている。このEITは、REIT命令などの命令実行によって発生するという点では例外であるが、その時実行中のコンテキストと関係なく起動されるという意味では割り込みである。つまり、例外と割り込みの中間的なものである。(IMASKフィールドを含むPSWはコンテキスト依存であるが、IMASKフィールドのみはコンテキスト独立として運用されるのが普通である。)詳しくは後の章を参照。

30 遅延コンテキスト例外(DCE)

Delayed Context Exception (DCE)

CSWレジスタ(あるいはDCEレジスタ)中のDCEフィールドが、PSW中のSMRNGフィールドよりも小さい値になった場合に発生する。この例外は、コンテキストに依存した各種の非同期の事象(入出力の完了など)を処理するために有効である。詳しくは後の章を参照。

【0421】[その他]

40 リセット割り込み(RI)

Reset Interrupt (RI)

外部からのリセット信号により発生する。

システムエラー例外(SEE)

System Error Exception (SEE)

EIT処理中に致命的なエラーが起きた場合に発生する。

[割り込み]

外部割り込み(EI)

50 External Interrupt (EI)

305

外部からのハードウェア信号により発生する。EITベクトルは外部から指定する。一般に、外部割り込みのチェックは命令の切れ目で行なわれるのが普通である。しかし、「本発明装置」の場合は、実行時間の上限の決まらない高機能命令（任意長ビットフィールド命令、ストリング命令、QSCH命令）が存在する。これらの命令では、命令の実行途中であっても外部割り込みを受け付けるようになっている。

固定ベクトル外部割り込み (FVEI)

Fixed Vector External Interrupt (FVEI) 10

外部からのハードウェア信号により発生する。EITベクトルは優先度毎に一つずつ決まっている。いわゆるオートベクトルの割り込みである。

【0422】以上のEITのうち、予約例外、不正例外、違反例外の区別は次のような考え方によっている。予約XXX例外機能拡張によって解消される可能性のあるもの将来の機能拡張によって、例外ではなくなる可能性がある。メーカー間のインプリメントの違いの出る場合がある。不正XXX例外意味的に明らかにエラーであるもの予約例外とは異なり、将来の機能拡張があっても、永久の例外のままである。メーカー間のインプリメントの違いは出ないところである。XXX違反例外リング保護の観点から実行を制限しているものその他OSやシステム構成上の例外、複数の分類に当てはまる例外な*

VS(Vector SM): EIT処理後のSM

ただし、VSがそのままEIT 処理後の SM となるのではない。詳細後述。

VX(Vector XA): EIT処理後のXA現在は0にreserved

(違反時は無視)

VAT(Vector AT): EIT 処理後のAT

VD(Vector DB): EIT処理後のDB

VIMASK(Vector IMASK): EIT 処理後のIMASK

ただし、VIMASKがそのままEIT 処理後のIMASK となるのではない。詳細後述。

VPC(Vector PC): EIT 処理後のPC

'=': 0にreserved (違反時は無視)

'=': 0にreserved

(違反時はシステムエラー例外)

プロセッサは、(EI)で生成したEITベクトル番号に従い、

(EITベクトル番号) × 8 + EITVTB

の物理アドレスにあるEITVTEを読み込む。

【0425】(E3) PSWの更新

読み込んだEITVTEをもとに、以下のようにPSWを更新する。

[外部割り込み以外の場合]

306

*ど

【0423】A9-2. EITのオペレーション

プロセッサがEITを検出すると、以下の手順にしたがってEIT処理を行なう。ただし、リセット割り込み(RI)、システムエラー例外(SEE)については、これとは違った動作をする。また、以下の説明は「本発明装置」32に限ったものであり、「本発明装置」64では、パラメータ等が異なってくる可能性がある。

(E1) ベクトル番号の生成

プロセッサはそのEITに応じたベクトル番号をプロセッサ内部で生成する。ただし、外部割り込みEIの場合は、プロセッサの外部(周辺LSIなど)からEITベクトル番号を得る。

【0424】(E2) EITVTEの読み込み

「本発明装置」では、それぞれのEITに対する処理ハンドラの先頭アドレスと、EITのベクトル番号との対応を示す表をEITベクトルテーブル(EITVT)と呼ぶ。また、そのテーブルの一つのエントリをEITVTEと呼ぶ。「本発明装置」のEITVTEは、EIT処理の自由度と拡張性を考慮して8バイトとなっており、EIT処理ハンドラの先頭アドレス(PC)だけではなく、PSWの一部のフィールドもセットすることができる。そのため、EITVTEはPC+PSWに準じた構成になっている。EITVTEのフォーマットは、図358のようにになっている。

min (VS, (旧SM)) ==> 新SM

スタックポインタの選択。

EIT発生前にSPI以外のスタックポインタを使っていた場合は、VSによってEIT処理ハンドラで使用するスタックポインタ(SPOまたはSPI)が選択される。EIT発生前に既にSPIを使っていた場合には、VSに関係なく、EIT処理ハンドラでもSPIをそのまま使う。このような仕様になっているのは、EITが

307

ネストした場合を考慮したためである。

旧RNG==> 新PRNG

00==> 新RNG

EIT処理ハンドラは、必ずリング0で実行される。なお、EITVTEには未使用のビットがあるので、将来的には、EITによってリング0以外のリングに入るような指定をすることも可能である。

VX==> 新XA

現在は0に固定されている。

VAT==> 新AT

EIT処理ハンドラの実行中は、アドレス変換の有無を切り換えることができる。

VD==> 新DB

EIT処理ハンドラの実行中は、デバッグの環境を切り換えることができる。

min(VIMASK, 旧IMASK) ==> 新IMASK

* min(VIMASK, 発生した外部割り込みの優先度) ==> 新IMASK

この部分のみ外部割り込み以外の場合とは異なっている。この機能により、優先度の低い多重割り込みを禁止することができる。なお、割り込みマスクの機能により、

発生した外部割り込みの優先度<旧IMASK
という関係が成立しているはずである。

【0427】(E4) プロセッサ情報のスタックへの退避

EIT発生前の旧PC、旧PSW、および発生したEITに関する各種の情報(EITNF-EITベクトルやスタックフォーマットなどを含む)をスタックに退避する。この退避に使用されるスタックは、新SMと新RNG(=00)により選択されるスタックである。この時に生成されるスタックフレームは、図359のようになる。このうち、EITINFは、発生したEITにより生成されるスタックフレームのフォーマット(FORMAT)、EITのタイプ(TYPE)、EITのベクトル番号(VECTOR)などの情報を32ビットに詰めたものである。追加情報の有無や内容は、EITの種類によって異なる。REIT命令では、EITINF中のFORMATを見ることによって追加情報の有無やそのフォーマットを知り、EIT発生前のもとの命令列に戻るための情報を得る。なお、「本発明装置」64になった場合に生成されるEITのスタックフレームは、旧PCで1ロングワード、旧PSWとEITINFを合わせて1ロングワード、という構成になる予定である。EITINFをPSWに隣接した場所に置いたのは、「本発明装置」64でのアラインメントの維持を考慮したためである。また、PSWをスタックトップに置いたのは、将来「本発明装置」64で、32ビットのコンテキストと64ビットのコンテキストが混在した場合でも、スタック中に退避されたPSW中のXAビットが読み出

308

*例外割り込みや内部割り込みによるEITの場合でも、EIT処理でIMASKを操作することができる。この機能を使えば、EIT処理に入ると同時に外部割り込みを禁止することができる。したがって、EIT処理と不可分に何らかの処理(例えばEITによって生成されたスタックフレームの転送など)を行ないたい場合に、この機能が有効である。

【0426】[外部割り込みの場合]

min(VS, IBSM) ==> 新SM

IBRNG ==> 新PRNG

00 ==> 新RNG

VX ==> 新XA

VAT ==> 新AT

VD ==> 新DB

せるようにするためである。

【0428】(E5) EIT処理ハンドラの起動

VPCをPCに転送し、EIT処理ハンドラを起動する。なお、命令プリフェッチにおいて発生したEITは、フェッチしようとした命令が必要になるまでEIT処理が遅らされる。これに対して、EIT処理ハンドラの最後に置かれたREIT命令では、次のような処理を行なってもとの命令列に復帰する。

【0429】(R1) スタックからの読み込み

スタックより、旧PSWとEITINFを読み込む。読み込んだPSW中のXAビットが0であれば、EITを発生したコンテキスト(タスクやプロセス)が32ビットのコンテキストであったということがわかるので、続いてスタックより32ビット幅で旧PCを読み込む。なお、「本発明装置」、32では、すべてのコンテキストが32ビットのコンテキストである。さらに、EITINF中のFORMATにより追加情報の有無を判定し、追加情報があれば、スタックからそれを読み込む。追加情報には、EXPC, IOINF, ERADDR, ERDATA, SPIなどがあるが、その詳細な意味はインプリメント依存である。FORMATがプロセッサのサポートしていない値(EITで発生するはずのない値)であった場合は、予約スタックフォーマット例外(RSFE)となる。

【0430】(R2) PSWの復帰

スタックから読み込んだ旧PSWにより、PSWの全フィールド(SMRNG, XA, AT, DB, IMASK, PSM, PSB)をEIT発生前の値に復帰する。この時、旧PSWがreservedの値を含んでいた場合には、予約機能例外(RFE)を発生する。

(R3) ストアバッファの再実行(インプリメント依存)

309

FORMATと追加情報の値によっては、REIT命令の中で、前回EITが発生したストアバッファによるライトサイクルの再実行を行なう場合がある。この時、ライトサイクルの実行に必要なアドレスとデータの情報としては、スタック上の追加情報の中にあったERADDRとERDATAが使用される。詳しくは、EITのタイプの説明の項を参照。なお、ストアバッファのみを再実行する機能があるかどうかということは、プロセッサのインプリメント上の問題である。すなわち、REIT命令のストアバッファ再実行機能を前提としてEIT処理が作られている場合には、当然REIT命令でそれをサポートする必要があるし、ストアバッファのみの再実行機能が無くても矛盾のないEIT処理が実現できていれば、REIT命令でそれをサポートする必要はない。プログラマから見た場合は、EIT処理とREIT命令による処理がきちんと対応しており、REIT命令によってEIT処理から戻った時に、EITが発生した命令列が矛盾なく実行を続けられれば良いのである。この機能の有無と、プロセッサとしての機能の高低とは直接関係しない。

【0431】(R4) EIT検出時に実行していた命令列への復帰

スタックから読み込んだ旧PCをPCに復帰し、PCの示す命令から実行を再開する。この時、EITINF中のTYPEフィールドによって、次に受け付けられるEITのタイプを変化させている。この機能は、多重EITの処理を矛盾なく行なうために、また命令のシングルステップ動作を、エミュレーションによる実行を含めて正確に行なうために利用される。なお、EITINFのVECTORフィールドは、REIT命令では特に使用されない。にもかかわらずVECTORがEITINFに含まれているのは、EIT処理ハンドラのプログラムに対して情報を提供するためである。

【0432】A9-3. EITのタイプ

「本発明装置」のEITを、EIT処理ハンドラ終了後の実行再開時のPCの位置と、EIT処理の優先度に着目して分類すると、以下のようになる。この分類は、EITINFのTYPEフィールドの値にそのまま対応する。

【命令中断型EIT (TYPE=0, PC不定)】このEITが発生すると、直ちにそれが検出され、EIT処理に入る。このEITが発生した場合、EITが発生した命令系列に復帰することはできない。これに該当するのは、RI, SEEである。

【命令完了型EIT (TYPE=1~3, PC次命令)】このEITが発生すると、その時実行中の命令処理が完了した後でそれが検出され、EIT処理に入る。一般には、このEITに対するEIT処理ハンドラの最後でREIT命令を実行することにより、EIT発生時に実行していた命令の次の命令から、命令の実行を再開

310

できる。なお、TYPE=1~3の区別は、優先度などの関係によるものである。これに該当するのは、TRAP, TRAPA, DBE, DI, DCEなどである。

【命令再実行型EIT (TYPE=4, PC現命令)】このEITが発生すると、プロセッサやメモリの状態は、EIT発生時に処理していた命令の実行開始前の時点に戻される。一般には、このEITに対するEIT処理ハンドラの最後でREIT命令を実行することにより、EIT発生時に実行していた命令から、命令の実行を再開できる。これに該当するのは、POE, ATR E, BAE, RIE, RFE, PIVE, IOEなどである。

【0433】命令完了型のEITは、以前実行していた命令に関するEITであり、命令再実行型のEITは、現在実行中の命令に関するEITである。したがって、複数のEITが同時に発生した場合には、一般に命令完了型のEITを先に処理する必要がある。また、命令中断型のEITは重要度の高いEITであり、これが検出された場合には、他のEITを処理することは無意味である。したがって、命令中断型のEITと他のEITが同時に発生した場合には、命令中断型のEITを先に処理する必要がある。結局、複数のEITが同時に発生した場合の優先度は

命令中断型>命令完了型>命令再実行型

となり、EITINFのTYPE=0~4が、そのままEITの優先度を表わすことになる。EITの種類とTYPEとの対応は、RI, TRAPのように明確に決まっているものもあるが、ある程度はインプリメント依存の部分がある。したがって、ソフトウェアでEITの要因を分析する際には、できるだけTYPEのフィールドを参照したり、書き換えたりしない方がよい。

【0434】例えば、ページ不在例外(POE)の場合、これは命令再実行型のEITであり、TYPE=4となるのが普通である。しかし、メモリの書き込みにストアバッファを用いるようなインプリメントのプロセッサにおいて、命令の最後の書き込みサイクル(ストアバッファ使用)でPOEが発生した場合には、この命令全体を最初から再実行しなくても、最後の書き込みサイクルのみやり直せば処理上の矛盾は生じない。そこで、このようなケースでのPOEを命令完了型のEITとし、エラーが発生した最後のライトサイクルの処理はREIT命令の中で行なうという場合がある。この場合は、POEでもTYPE=1となる。また、EIT処理でスタック積まれるPCは、POE発生命令ではなく次の命令になる。命令再実行方式に忠実にしたがう限り、命令実行中にエラーが発生した場合は、命令実行前の状態に戻してTYPE=4のEITを起動するというのが原則である。しかし、命令がもう少しで終了するということでエラーが発生した場合には、一応命令を終わったこととしてTYPE=1のEITを起動し、残りの処理(ス

311

トアバッファのライトサイクル)はREIT命令に任せるというインプリメントも可能なのである。このようなインプリメント方法をとるのであれば、POEのTYPEが1と4の2通りになる。この場合、TYPEの違いによってREIT命令で必要となる処理が変わってくるので、REIT命令はそれに対応できるようになっていなければならない。この方法では、命令最後のライトサイクルで発生したエラーによるEITに対して、命令全体を再実行するのではなく、最後のライトサイクルのみを再実行するという形になっている。つまり、一種の命令継続実行方式になっている。この場合、命令継続実行*

PSW : EITを検出した時点のPSW

Format: スタックフォーマット番号(8bit)

Type : EITタイプ(8bit)

Vector: EITベクトル番号 (9bit)

PC : EITハンドラからの復帰後の、実行再開アドレス

この内、「その他の情報」は各EITのスタックフォーマット番号に応じて異なり、EITの要因を解析するための情報、EITハンドラから復帰するための情報が含まれる。

PC : REIT命令でEITから戻った後に実行する命令の先頭アドレス

ドレス

EXPC : EITの検出時に実行していた命令のPC。

IOINP : 入出力に関する情報

Error Addr: EITを発生させたバスサイクルのアドレス

Error Data: EITを発生させたバスサイクルのデータ (writeのみ)

SPI : EIT検出時のSPIの値

【0436】フォーマット番号0: 予約命令例外、予約機能例外、予約スタックフォーマット例外、リング遷移違反例外、〈〈L1〉〉命令例外、コプロセッサ命令例外、特権命令違反例外、不正オペランド例外、固定ベクトル外部割り込み、遅延割り込み例外、外部割り込み。フォーマット番号1: バスアクセス例外、アドレス変換例外。

フォーマット番号2: デバッグ例外、奇数アドレスジャンプ例外、ゼロ除算例外、条件トラップ命令、トラップ命令。

フォーマット番号3: DBG外部割り込み、DBGトラップ命令、DBGデバッグ例外、DBGアクセス違反例外 (つまりDBG EIT専用)

【0437】EXPCは次のような目的で導入されたものである。

・エラー解析情報の提供

—ストアバッファの書き込みの際にTYPE=1のEITが発生したような場合、その書き込みを行った命令を指しているのがEXPCである。PCは先に進んでしまっている。

—デバッグ例外では、PCは次の命令を指し、EXPC

312

*のために退避される内部情報に相当するのが、EITの追加情報としてスタック上に退避されるERADDRやERDATAである。なお、ストアバッファを用いた書き込みサイクルで発生するEITの場合、その書き込みを行なった命令の直後にEIT処理に入るとは限らない。

【0435】A9-4. EITのスタックフォーマット
EITの検出にともなって、EIT処理に必要な情報がスタックに退避される。そのスタックフォーマットは図360の通りである。

※まれている。スタックフォーマット番号に応じたスタックフォーマットを図361に示す。

30 は前の命令を指している。したがって、例えば、ジャンプ命令実行時にデバッグ例外を起動するようにした場合、EXPCによってジャンプ前のPCの値を、PCによってジャンプ後のPCの値を知ることができる。

・多重EITの処理

—TYPE=1のTRAPAなどのEITの場合は、その処理ハンドラの中でEXPCの情報が必要になることはない。しかしながら、TYPE=1のEIT (TRAPAなど)とTYPE=2のEIT (デバッグ例外など)が同時に発生した場合、TYPE=1のEITにおいて、TYPE=2で使用するEXPCを退避しておかなければならない。TRAPAでもEXPCを退避するような仕様になっているのは、このためである。この場合、TRAPAの処理に対するREIT命令実行後のEXPCは、REIT命令の先頭を指すのではなく、スタックからポップした旧EXPCの値を復帰したものになっていなければならない。すなわち、REIT命令の直後になっていたデバッグ例外を起動した直後にペンディングになっていたデバッグ例外を起動した場合、スタックに退避されるEXPCは、REIT命令を指すのではなく、TRAPA命令を指すものでなければならない。

313

314

(なお、この例は、TRAPAのEITVTEでデバッグ例外をマスクすることを想定したものである。)

【0438】また、IOINFの構成は図362のようになっている。

= : '0'にreserved

WI : REIT 命令におけるライトリトライの指示

メモリアクセス系のEIT

(TYPE=1) で意味を持つ。

WI=0 ライトリトライ要

WI=1 ライトリトライ不要

MEL : アドレス変換例外の発生位置

0000 エラーなし

0001 アクセス権に関するエラー

0010~1110 (reserved)

1111 I/O領域に関するアクセスエラー

MEC : メモリアクセス関係のエラーのエラーコード

0000 エラーなし

0001 未使用領域参照エラー

0010 (reserved)

0011 (reserved)

0100 readに関するリング保護違反エラー

0101 write に関するリング保護違反エラー

0110 execute に関するリング保護違反エラー

0111 (reserved)

1000 read時のバスアクセス不能

1001 write 時のバスアクセス不能

1010 (reserved)

1011 (reserved)

1100 (reserved)

1101 I/O 領域に対するメモリ間接アドレッシング

315

316

1110 I/O 領域に対するexecute
 1111 read時にI/O 領域とI/O 以外の領域をまたぐ
 write 時にI/O 領域とI/O 以外の領域をまたぐ

RW : バスサイクル種別

RW=0 write

RW=1 read

BL : バスロック状態

BL=0 バスロック中でない

BL=1 バスロック中

PA : 空間の指定

PA=0 (reserved)…論理空間(アドレス変換あり)

PA=1 物理空間(アドレス変換なし)

AT : EITが発生したバスサイクルのアクセスタイプ

AT=000 データ

AT=001 プログラム

AT=010 割込みベクトルフェッチ

AT=011~111 (reserved)

SIZ : ライトリトライを行う際のデータサイズ

0000 (reserved)

0001 1 バイト

0010 2 バイト

0011 3 バイト

0100 4 バイト

0101~1111 (reserved)

【0439】A9-5. 「本発明装置」のEITベクトル・テーブル

図363、図364に示す

リセット割込みとDBGモードのEIT (No. 0~5) に関するEITテーブルのエントリはSPI値とPC値で構成される。その他のEITに関するEITテーブルのエントリはPSW値とPC値で構成される。EITVTEのリセット時の初期値は'FFFFFF000'であるため、リセット割込みでは物理番地の'FFFFFF000'からエントリ(SPI, PC)がフェッチされる。

【0440】A9-6. EIT処理中のエラー

EIT処理中(EIT発生から状態の退避を経て新PSWの設定まで)に、別のEITが発生するような重大なエラーが起きた場合には、システムエラー例外(SEE)となる。システムエラー例外(SEE)となる可能性があるのは、EITVTEの読み込みに伴うバスアクセス例外、旧PC、旧PSWの退避に伴うスタックのページ不在例外、アドレス変換例外などである。また、EITVTEのVPCを含むワードのLSBが'1'であった場合にも、システムエラー例外(SEE)とな

る。システムエラー例外(SEE)の発生は、SPI, SPOのどちらかのスタックを使用するかということには関係しない、SPOのスタックでページ不在例外が起った場合にも、SPIのスタックに切り換えてあるいはページ不在例外のEITVTEで指定されるスタックに切り換えて)、EIT処理中を継続するという仕様になっていない。一方、JRNGによるリング遷移はEITでないので、JRNGの処理中にページ不在例外が起った場合には、ページ不在例外のEITVTEで指定されるスタックを使って、ページ不在例外のEIT処理中を行うことになる。この点で、EIT処理中に含まれるTRAPAとEIT処理中に含まれないJRNGでは、システムエラーとなるまでのステップが一段異なっているので、注意が必要である。(図365参照)

いずれにしても、OSを作る場合には、SPIにより指定されるスタック領域はメモリ常駐とし、SPOにより指定されるスタック領域も、特殊な使い方をする場合を除けばメモリ常駐となるようにプログラミングする必要がある。

【0441】A9-7. 多重EIT

TYP=0のEITを除けば、EITの検出とそれに対

317

する処理は、各命令の切れ目で行われる。したがって、場合によっては、命令の切れ目において複数のEITが同時に検出される可能性がある。これを多重EITと呼ぶ。ここでは、多重EITの処理順序について説明する。例えば、TYP=0のTRAPAとTYP=3の外字割り込み(EI)が同時に発生した場合、まずTRAPAに対するEIT処理中が行なわれ、引続きEIに対するEIT処理中が行なわれる。その結果、PC、PSWとスタックの状態は図366のようになる。したがって、この例では、EIT処理中終了後にまずEIの処理ハンドラが実行される。EIの処理ハンドラが終了した後は、その最後に置かれたREIT命令により、一段浅いレベルにあるTRAPAの処理ハンドラに移る。つまり、優先度の高いTRAPA処理ハンドラの方が後回しになるわけである。ただし、上の例ではTRAPAのEIT処理中が先に行われるため、そこでPSWを変更してEIをマスクすることができる。つまり、TRAPAのEITVTEで

VIMASK < EIの優先度

となるような指定を行っておけば、TRAPAのEIT処理中でIMASKが変更され、EIに対するEIT処理中は行なわれなくなる。この場合は、TRAPAの処理ハンドラが実行される。そして、ハンドラの最後のREIT命令でIMASKが元の値に戻った時に、マスクされていたEIが起動されることになる。このように、優先度の高い(TYPEの小さい)EIT処理中におけるPSWの更新によってマスクされるEITには、DBE、EI、DI、DCEのEIT、つまりTYP=2~3のEITがある。逆に言えば、マスク可能なEIT

(処理要求を保持することが可能なEIT)が、優先度の低いTYP=2~3になっているわけである。これに対して、TRAPAの場合は、EIT処理中要求を保持するレジスタやハードウェアは何も用意されていない。PCも次の命令に進んでいるので、TRAPA命令を再実行することもできない。したがって、TRAPA命令実行直後にEIT処理中を行わないと、EIT処理中要求が失われてしまう。TRAPAをTYP=1の高い優先度に行っているのは、このためである。また、TYP=4のEITは命令再実行のEITであるため、他のEITに対する処理が終わった後でもう一度同じ命令を実行すれば、再び同じEITが発生する。命令実行型(TYP=4)のEITが最も低い優先度になっているのは、このためである。したがって、多重EITの場合には、TYP=4のEITの処理は行う必要がない。TYP=4のEIT起動要求は、同時に発生しTYP=1~3のEITの検出に伴ってキャンセルされる。REIT命令実行直後に受け付けられるEITとは異なっている。REIT命令では、スタック中からポップされたEITINFの中のTYPEによって、REIT命令終了直後に受け付けるEITを調整している。REIT命令実行後

318

に受け付けられるEITのTYPEは図367の通りである。このうち、TYPE=2はデバッグ例外(DBE)である。つまり、デバッグ例外に対するEIT処理中ハンドラのREIT命令実行直後には、デバッグ例外を受け付けられないということを意味している。

【0442】REIT命令実行直後かどうかによって、TYPE=2のデバッグ例外の扱いが異なっているのは、1命令毎のシングルステップ実行を行うためである。この場合、デバッグ例外に対するREIT命令の直後で再びデバッグ例外を起こしていたのでは、被デバッグプログラムの実行が全く進まずに、デバッグ例外のみが続いて発生するという状況になってしまう。したがって、上記メカニズムにより、REIT命令の直後ではデバッグ例外を発生せず、1命令実行してからデバッグ例外を発生するようにしているのである。一般に、シングルステップ実行を行う場合には、次に命令を実行するか、デバッグ例外を起動するかとの二つの内部状態を持つ必要がある。「本発明装置」では、この二つの状態を、REIT命令実行直後かどうかといった内部状態と、EITのTYPEとの組み合わせによって表現していると考えられる。なお、この考え方によるシングルステップ実行は、デバッグ例外と同時にほかのEITが発生した場合にも適用できる。予約命令例外(RIE)のEIT処理中ハンドラで命令エミュレーションを行う場合は、他のEIT(ページ不在など)に対する処理ハンドラとは異なり、RIEの処理ハンドラの前後でデバッグ例外を起動しなければならない。例えば、シングルステップ実行後に通常命令→デバッグ例外→ページ不在例外であれば次は通常命令を実行する必要があるが、通常命令→デバッグ例外→予約命令例外(エミュレーション)であれば次はデバッグ例外の起動となる。これは、ページ不在例外がデバッガやデバッグ対象プログラムにとって全く見えないものであるのに対して、エミュレーション例外は、デバッガ対象プログラムにとって「一命令の実行」として見えるはずのものだからである。「本発明装置」の場合は、予約命令例外のEIT処理中ハンドラの中でEITINFのTYPEを調整することにより、以上のようなことが可能である。

【0443】A9-8. 「本発明装置」のDI

A9-8-1. DIのオペレーション

「本発明装置」のDI(Delayed Interrupt)は、DIレジスタ中のDIフィールドがPSW中のIMASKフィールドよりも小さい値になった場合に発生するEITである。この機能は、コンテキストとは独立した非同期の事象をペンディングとして処理要求だけを登録したり、処理順序をシリアライズしたりする時など有効なものである。DI処理のEITベクトルは、各割り込み優先度毎に15種類用意されている。IMASKの値と、そのフラグ変化のその時に許可される外部割り込みとの関係は、図368のようになっている。

319

る。DIを起動するかどうかのチェックをする必要があるのはIMASKが大きくなった時、またはDIが小さく

```
LDC src,@psw ;pswは制御空間中のPSW のアドレス
LDC src,@imask ;imaskは制御空間中のimask のアドレス
LDC src,@DI ;DI は制御空間中のDIのアドレス
REIT
WAIT
```

の各指令である。このうち、LDC src,@di以外の場合には、これらの命令を実行する前のDIフィールドの値が、起動されるDIのレベル（優先度）となる。DIのレベルは、DIとして起動されるEITのベクトル番号に影響する。また、LDC src,@diによってDIが起動された場合には、LDC実行前のDIフィールドの値ではなく、LDCによって新しくセットされたDIフィールドの値（つまりsrc）が起動されるDIのレベルとなる。なお、EIT起動時（外部割り込み、例外、TRAPすべて含む）にもIMASKの変化するところがあるが、この場合にはIMASKの値が大きくなることはないため、DIは起動されない。DIが起動された場合、DIフィールドは1111（要求なし）にリセットされる。また、IMASKフィールドは、受理されたDIのレベルを優先度とする外部割り込みが発生したのと同じ変化をする。つまり、min(VIMASK, 受理されたDIのレベル) ==> 新IMASKとなる。

※ I は実行中の状態を表し

【0446】一般のシステムコール処理

図369に示す。

外字割り込み処理ハンドラからのシステムコール

図370に示す。DIの機能を使えば、本発明装置の遅延ディスパッチの処理をすなおに実現することができる。また、多重割り込みやシステムコールのネストが起こった場合にも容易に対処できる。

【0447】A9-9. 本発明装置のDCE

A9-9-1. DCEのオペレーション

本発明のDCE (Delayed Context Exception) は、DCEレジスタ（またはCSWレジスタ）中のDCEフィールドよりも小さい値になった場合に発生するEITである。この機能は、コンテキストに関連した非同期の事象（入出力の完了など）の処理をペンディングとして処理要求だけを登録したり、処理順序をシリアライズしたりする時などに有効なものである。DCEレジスタ（またはCSWレジスタ）中のDCEフィールドは、DCE要求を入れるためのフィールドは、DCE要求を入れるためのフィールドである。DCEレジスタ（またはCSWレジスタ）はコンテキスト毎に固有のレジスタであるため、コンテキスト毎に別々

320

* くなった時である。したがってこれに該当するのは、

※【0444】A9-8-2. DIの使用例

10 【例；本発明装置の遅延ディスパッチ (delayed dispatch)】 本発明装置では、外部割り込み処理ハンドラの中から発行したシステムコールによってレディキューの状態が変わった場合に、それに伴うディスパッチング（レジスタの入れ替えなど）が割り込み処理ハンドラから戻るまで遅らされる。これは、多重割り込みに伴う矛盾を避けるためである。これをDIの機能によって実現する。

前提条件

20 ・システムコールはTRAPAのEITVIEでは、VIMASKでは、VIMASK=14を指定しておく。これは、DI機能によって、システムコール処理の最後のディスパッチングを行うためである。

・ディスパッチングの処理を行う部分は、DI14によって起動される。

【0445】

【数18】

※ は実行を中断した状態を表す。

のDCE要求を与えることが可能である。DCEは各コンテキストに付随したものであるから、コンテキストとは独立した外部割り込みの処理中（SM=0の場合）には、DCEは起動されない。また、他のコンテキストAでより高い優先度のDCE要求が出ていても、そのコンテキストAにディスパッチされない限り、コンテキストAのDCEは起動されない。別のコンテキストBで出ているDCE要求がそれより低い優先度であったとしても、コンテキストBに先にディスパッチされれば、コンテキストBのDCEが先に起動される。DCEフィールドの値と、その時に起動されるDCEとの関係は図371のようになっている。いずれの場合にも、SMRNG>DCEとなった時にDCEが起動される。(reserved)の指定をした場合、実際にはDCE=000と同じ動作をする。ただし、将来の拡張のため、この機能を利用したプログラミングを行なってははいけない。

【0448】DCEの起動される可能性があるのは、SMRNGが大きくなった時、またはDCEフィールドの値が小さくなった時である。したがって、この条件に該当する以下の命令において、DCEを起動するかどうかのチェックをする必要がある。

321
 LDC src, @psw ; psw は制御空間中のPSW のアドレス
 LDC src, @smrng ; smrng は制御空間中のSMRNG のアドレス
 LDC src, @dce ; dce は制御空間中のDCE のアドレス
 LDC src, @csw ; csw は制御空間中のCSW のアドレス
 ; ただし、CSW は実装されない場合がある。

REIT

RRNG

なお、EIT起動時（外部割り込み、例外、TRAPすべて含む）やJRNG実行時にもSMRNGの変化することがあるが、EITやJRNGの場合はSMRNGの値が大きくなることはないため、DCEが起動されることはない。DCE自体は、一つのEIT処理として起動される。DCEのEITが起動された場合、DCEフィールドは111（要求なし）にリセットされる。SMRNGフィールドは、一般のEIT処理と同じように、DCEのベクトル番号に割り当てられたEITVTEにしたがって変化をする。DCEはコンテキスト毎の処理であるため、起動されたEIT処理ハンドラでは、SPIではなくSPOを使用するのが普通である。EITVTEの設定によっては、DCE処理でSM=0（SPI使用）に入ることも可能であるが、これは運用上の問題として対処し、ハードウェアでは特にチェックは行なわない。REIT命令やRRNG命令によつてDCEが起動された場合、実際にDCEを起動する処理はREITやRRNGと同時にこなされてもよいが、動作仕様上は、一旦REITやRRNGの実行が終わってからEITを起動するという形になる。たとえば、DCE=110の*

〔DCE 要求設定時〕

if (DCE=111) then

新しいDCE 要求=> DCE フィールド/*DCE要求これだけ*/

else

新しく発生したDCE 要求を、リング順に構成されたDCE 要求キューに入れる

endif

〔DCE 処理時〕

/*DCE起動の際、ハードウェアにより111=>DCE となる*/

if (DCE 要求キューが空でない) then

DCE 要求キューの次のエントリをDCE フィールドにセット

endif

【0450】A9-9-3. DCEの使用例

〔例；入出力管理プログラムの起動〕外部割り込みによつて入出力完了が通知され、それによつて、プロセスAの入出力管理部（ring1）がプロセスAに対して非*

は実行中の状態を表し、は実行を中断した状態を表す。

【0452】①の開始アドレスはプロセス（コンテキスト）毎に指定されるべきものだが、実際にはDCEのEIT処理ベクトルがプロセス間で共通であるため、OS

10 *時にRRNGでring1からring3に戻ると、そこでDCEが起動されてring0に入る。この時、PRNGはring1ではなくring3となっていなければならない。DCEとDIや外部割り込みを比較すると、図372のようになる。入出力の完了通知などの場合には、外部割り込み処理ルーチンの中で、該当するコンテキストDCEを起動するという流れになることもある。DCEをソフトウェアによつてシミュレーションすることは不可能ではないが、一般には、スタック上に退避されたPSWやPCの変更をしなければならないため、かなり面倒である。割り込んだプログラムが、割り込まれたプログラムのスタックフォーマットをすべて知っていなければならないからである。

【0449】A9-9-2. DCEのネスト

DCEは多重ネストができるとより効果の大きいものになる。したがって、DCE要求が複数発生した場合に、どのような処理をするかが問題となる。本案装置では、ネストの処理（要求のキューイング）はソフトウェアで行なう方針である。

<<複数のDCE要求のキューイング処理例>>

※同期に起動されるものとする（図373参照）。

【0451】

【数19】

は実行を中断した状態を表す。

でプロセス毎のDCE要求テーブルを解析し、そこへジャンプする必要がある。この図の場合は、外部割り込みが発生した時にたまたまプロセスAが実行中だったわけ

323

である。他のプロセスの実行中に入出力の外部割り込みが発生した場合、ring1の入出力管理部の起動は、プロセスAへのディスパッチが行なわれるまで遅延させられる。

・ 0 にreserved (違反時例外発生)

・ 1 にreserved (違反時例外発生)

このビットが0(1)であれば処理が正常に行なわれ、このビットが1(0)であれば予約命令例外(RIE)を発生する。

・ 0 にreserved (違反時も無視)... Ver0.87 の *

・ 1 にreserved (違反時も無視)

ユーザへのマニュアルには、将来の拡張のためこのビットを0(1)にしておくように明記する。実際には、このビットが0(1)であっても1(0)であっても動作は同じである。

「違反時も無視」というのは、アーキテクチャ上あまり好ましいことではないが、命令ビットパターンの割り当て、将来の拡張性、命令の高速実行の上からやむを得ない場合がある。

・ 0 にreserved (違反時動作を保証しない)

・ 1 にreserved (違反時動作を保証しない) ユーザへのマニュアルには、将来の拡張のためこのビットを0(1)にしておくように明記する。実際には、このビットが0(1)の場合は正常な動作を行なうが、1(0)の場合の動作はインプリメントに依存して異なっている。

「違反時動作を保証しない」というのは、アーキテクチャ上あまり好ましいことではないが、インプリメントや命令ビットパターンの割り当て

、命令の高速実行の上からやむを得ない場合がある。例えば、LDATE.MU LXの第一ハーフワードの'R'がこれに該当する。

324

*【0453】付録10. 本発明装置の命令ビットパターン

〔表記法に関する注意〕 命令ビットパターンの表記は、次のように行なう。

【0454】A10-1. 命令のフォーマット別ビット割り当て

〔ビット割り当てに関する注意〕

・本発明装置では使用できるアドレッシングモードが命令毎にかなり異なっており、そのチェックをする必要がある。チェックを容易にするため許されるアドレッシングモードが区別しやすいようにビットパターンの割り当てを行なう。特定のアドレッシングモードを禁止しているオペランドの場合は、原則としてそれがそのオペランドを含むハーフワードだけで分かるようになっている。

・Pビットは、原則としてオペランド毎(レジスタ直接指定とイミディエート指定を除く)、および暗黙のスタック参照について一つずつ独立に入れる。命令パターン中では、'P'または'Q'で表す。ただし、一般形の命令でカバーできる場合は、同じ命令の短縮形ではPビットが入らないこともある。(PUSH, POP, PUSH SHAのみは、スタック参照に関するPビットがない。)

・ビットパターンのうちLV reservedで示されているものは、各メーカーで自由に使用してよい命令ビ

ットパターンである。この命令ビットパターンは、例えばICEとのインターフェースを行なうための、ユーザに解放しない命令として利用することができる。以下図374~図385に示す。

【0455】A10-2. 予約命令例外の検出について 上記のビットパターン中で{RIE}で示されたパターンは、将来の拡張のために予約された(reserved)のビットパターンである。この命令ビットパターンを実行すると、予約命令例外(RIE)は発生する。このほか、インプリメントされていないオプションやサイズ(未実装の<<L2>>)を含むを指定した場合、未定義のオプションを指定した場合、命令ビットパターンの'-'の部分で'1'にした場合、命令ビットパターンの'+'の部分で'0'にした場合、命令中の'P', 'Q'のビットを'1'にした場合、reservedの条件(cccc)や終了条件(eeee)を指定した場合にも、すべて予約命令例外(RIE)となる。現在、LDATE, MULXなどの例外を除けば、原則として第一~第四バイトについてはすべての命令パターンをチェックし、パターンが違っている場合には、

325

R I E にしている。第五、第六バイトについてはチェックを行わず、パターンが違ってもエラーとはしていない。上記のビットパターンのうちで、特に { R I E - X } で示したものは、第一HWが汎用アドレッシングモードを含み、第二HWまで読まないで R I E であることがわからないビットパターンである。この場合は、第一HWの E a の拡張部の後に第二HWが置かれる。現在使用していないビットパターンのうち、将来の機能拡張にともなって実装の予定のあるパターンや、他のメーカーのチップと動作の異なりそうなパターンについては、特に積極的に検出を行なうべきである。これは、その命令パターンを実行した場合の間違いを防ぐためである。このような方針に基づいた場合、予約命令の例外 (R I E) のチェックの優先度は次のようになる。

【0456】↑高優先度

(既に意味が決まっているもの)

未定義の < L 2 > 機能の指定

64ビットサイズの指定 (R R , M M , W W , S S = 1)

(命令の拡張に利用される可能性の高いもの)

{ R I E } となっている命令パターンの指定

B V P A T ~ B V S C H の ' + X ' の ' + '

P S T L B ~ E X I T D : G のグループの第二HWの ' - ' P ビット指定

(命令の拡張にはまず利用されないもの)

L D A T E ~ I N D E X のグループの第一HW ' ! R ' の ' ! '

S T A T E ~ Q I N S のグループの第二HW ' + W ' の ' + '

P S T L B ~ E X I T D : G のグループの第一HW ' + X ' の ' + ' 30

A C B : R , S C B : R の第二HWの ' - '

↓低優先度

現在チェックすべきビットパターンは前述のような仕様

になっているが、今後、このような方針に基づいて予約 *

' - ' は 0 にリザーブされるビット

** < R n > 0 * * X X * * * * R n がインデクス

** - 0 1 * * - * * * * インデクスなし

** - 1 1 * * - * * * * R C がインデクス

X X ≠ 0 0 によるスケーリングは利用できない

***** ** 0 < d 4 > 4 ビットディスプレースメント

***** ** 1 - 0 1 16 ビットディスプレースメント

***** ** 1 - 1 0 32 ビットディスプレースメント

***** ** 1 - 1 1 64 ビットディスプレースメント

< d 4 > のサイズ指定部分と M I S C モードの d i s

p : 1 6 , d i s p : 3 2 の指定部分が同じビット位置

* 命令例外の検出に関する詳細仕様の調整を行ない、一部は変更の行われる可能性がある。なお、命令をどこまで読んだ時に E I T を起動するかということは、特に規定しないものとする。第一HWだけで E I T を起動することが明らかである場合にも、第二HWまで読んでもよい。また、オペコード部だけで E I T を起動することが明らかである場合 (予約命令例外など) に、E a の拡張部まで処理しても構わない。

【0457】A 1 0 - 3 . オペランドフィールド名索引
図 3 8 7 , 図 3 8 8 に示す。

【0458】A 1 0 - 4 . アドレッシングモードのビット割り当て

共通ビットパターン

・サイズ関係

0 1 16 bit

1 0 32 bit

1 1 64 bit

・アドレッシングモード

0 0 0 r e g + など

0 1 16 bit 相対間接モード

1 0 32 bit 相対間接モード

1 1 付加モード

・レジスタ指定

0 0 (特殊)

0 1 (S P)

1 0 a b s または 0

1 1 P C

【0459】付加モード

E I < R x > M S P X X D < d 4 >

になっている。

【046.0】基本モード

327

328

P000 xxxx MISC P=0:Sh
 0000 {R1E}
 0001 {R1E}
 0010 {R1E}
 0011 {R1E} - @ads:64
 0100 @SP+(read:@sp+, write:illegal, rmw:illegal)
 0101 @SP+(write:illegal, read:@sp+, rmw:illegal)
 0110 {R1E}
 0111 {R1E}
 1000 {R1E}
 1001 @ads:16
 1010 @ads:32
 1011 絶対付加モード
 1100 imm+(read:@sp+, write:illegal, rmw:illegal)
 1101 @(disp:16, Rn)
 1110 @Rn

1111	@(disp:32, Rn)		[Ea]	[Sh]
0001 <Rn>	Rn	Sh	0000 00**	00 00**
1001 xxxx	{R1E}		0000 011*	00 011*
P010 <Rn>	@(disp:16, Rn)	P=0:Sh	0000 1000	00 1000
P011 <Rn>	@Rn	P=0:Sh	0101 **** (<<L2>>未実装時のみ)	
P100 <Rn>	@(disp:32, Rn)		0111 **** (<<L2>>未実装時のみ)	
P101 <d4>	@(disp:4, FP) <<L2>>		1*** ****	
P110 <Rn>	レジスタ相対付加モード			
P111 <d4>	@(disp:4, SP) <<L2>>			

・***1 ****の 패턴の時には拡張部が付かない。

・@ads:64の割り当てが変則的であるが、64ビット拡張部に対しては、おそらく内部の回路でも特殊な扱いが必要であると思われる（例えば、本発明装置でも @ (disp:64, Rn) は基本モードで実現できない）ので、それほど問題ではないと思われる。むしろ、付加モードとの関係が統一的になるようにした。

【0461】未定義のアドレッシングモードを指定した場合（EA中のPビット=1を含む）には、予約命令例外（R1E）となる。具体的には、以下のパターンの場合にR1Eとなる。

30 付加モードでリザーブのパターンを指定した場合にも、予約命令例外（R1E）となる。M=1で<Rn>≠0000, 0001の場合、D=1で<d4>≠0001, 0010以外の場合、P=1の場合、XX=11の場合もこれに含まれる。付加モードのある段で、PCに対して、×2、×4、×8以外のスケールを指定した場合には、その段の終了処理後の中間値として、インプリメントに依存した不定値が入る。EITにはならない。(<<L2>>)未定義で、5段以上の付加モードを指定した場合にも、予約命令例外（R1E）となる。〔詳細調整中。予約機能例外RFEとなる可能性もある。〕命令によって使用できないアドレッシングモードの組み合わせを指定した場合（JMP #imm_data, CMP #0, #1など）にも、予約命令例外（R1E）となる。(<<L2>>)未実装のために実行できない組み合わせを指定した場合も、これに含まれる。（これに当てはまるものは、レジスタ指定のビットフィールド命令である。）

【0462】A10-5. 命令オプションのビット割り当て

50 いずれの場合にも、最初に記述した方（オプション値が

329

330

0, 00...の方) がアセンブラでのデフォルトになる。

cccc Bcc.TRAP/cc での条件指定

eeee スtring命令、QSch命令での終了条件指定

p,q... P ビット指定(Q...は、P ビットの必要なオペランドが複数の場合)

b /F=0,/B=1(BSCH,BVSCH,BVMAP,BVCPY,SCMP,SMOV,QSCH)

r /F=0,/R=1(SSCH)

c /N=0,/S=1(CHK)-CHK,change index valueの 'c'

d /0=0,/i=1(BSCH,BVSCH) -dataの 'd'

m /NM=0,/MR=1(QSCH) -maskの 'm'

p /AS=0,/SS=1(PTLB,PSTLB,LDATE) -PTLB,specific space の 'p'

ttt /PT=000,/ST=001,/AT=110, {RIB} =010~101,111(PSTLB,LDATE,STATE)

xx /LS=00,/CS=01, {RIE} =10,11(LDCTX,STCTX)

【0463】A10-6. Bcc命令、TRAP/cc 命令の条件指定 (cccc)

ccccの値の割り当ては、図389のようになる。

【0464】A10-7. 終了条件の指定 (eeee)

eeeeの値の割り当ては、図390のようになる。0

000~0101については、cccc (Bcond命令

の条件指定) のビットパターンに意味を合わせてあ

る。特に、LTU, GEUは、SUBX命令などにおい

て、オペランドを符号なしデータと考えた場合の比較結

果がX_flgに反映されるのに合わせたものであ

る。なお、(L2)の終了条件のうち、二つの条件

が、or. で結ばれているものについては、そのうちの

いずれの条件で終了したかを示すために、M_flg

を使用する。M_flgがセットされるのは、原則と

して R4との比較によって終了した時であり、具体的

には図391のような場合である。M_flg=1の

条件を満たさなかった場合、および、これ以外の終了条

件で終了した場合には、M_flg=0となる。

(L2)の終了条件をインプリメントしない場合に

は常にM_flg=0となるが、その場合でも、将来

はM_flg≠0となる場合が生じるということを、

マニュアルに明記しておくのが望ましい。

【0465】A10-8. BVMAP命令の演算コード

R5の下位4ビットに入れる演算コードである。これを

図392に示す。

【0466】A10-9. アドレッシングモード対応

各命令のオペランドと、禁止されているアドレッシング

モードモードとの対応を図393~図395に示す。○

の組み合わせに対しては、そのアドレッシングモード

モードが使用可能である。×の組み合わせに対しては、そ

れを実行しようとした場合に予約命令例外 (RIE) が

発生する。

【0467】付録11. 高機能命令の詳細仕様と終了時

のレジスタ値

各命令の解説の項では、高機能命令の詳細や終了時のレ

ジスタ値について明確に述べられていないので、ここで
まとめて説明を行なう。

【0468】A11-1. 高機能命令の仕様決定の方針

SMOV/B, SCMP/B, BVMAP/B, BVC

PY/Bでは、@-SPとの対応などからプリデクリメ

ントの形で処理を行なうという考え方と、SMOV/

F, SSCH/Rなどとの整合性からポストデクリメ

ントの形で処理を行なうという考え方とがある。例えば、

H'100~H'1ffの領域をSMOV/B. Bによ

って転送する場合、SMOV/Bがプリデクリメントの

仕様であればレジスタの初期値はH'200となるし、

SMOV/Bがポストデクリメントの仕様であればレジ

スタの初期値はH'1ffとなる。

[ポストデクリメントのデメリット] SMOV/FとS

MOV/B, SCMP/FとSCMP/Bとの対称性が

悪くなる。例えば、H'000000ffまでの領域を

占めるstringに対してSMOV/Bを実行する場

合、SMOV/B. Bであればポインタの初期値として

H'000000ffを設定し、SMOV/B. Wであ

ればポインタの初期値としてH'000000fcを設

定する必要がある。

【0469】[プリデクリメントのデメリット] SSC

H, BSCHなどのサーチ系の命令との整合性が悪くな

る。もし、SSCHで、命令終了後のポインタ最終値が

必ず終了条件成立のエレメント (サーチ結果のエレメン

ト) を指すという原則を設けるとすると、/F, /B,

/Rといった処理方向によってプリ更新/ポスト更新を

変えることはできなくなる。したがって、/Bのみプリ

デクリメントとするわけにはいかない。(実際にはSS

CH/Bは存在しないが、BSCH/Bなどの仕様との

関連がある)

TRONCHIPでは、[ポストデクリメントのデメリ

ット] の方を重視し、SMOV/B, SCMP/Bでは

プリデクリメントの仕様にする。次に、SMOV, SC

MP, SSCHが終了条件によって終了した場合、ポイ

331

ンタの更新を行ってから命令を終了するか、ポインタの更新前に命令を終了するか、という問題がある。

【0470】[ポインタの更新前に命令を終了するデメリット] エレメントサイズによって命令を終了する場合には、ポインタの更新が行なわれ、ポインタが次のエレメント（/Fの場合はまだ処理の終わっていないエレメント）を指すようになってから命令を終了するので、その仕様とは合わなくなる。つまり、終了条件が成立するかどうかによってポインタを更新して良いかどうかの状況が変わるため、仕様がわかりにくくなる上、高速なインプリメントが難しくなる。また、SSCHで、サーチが成功してから連続して次のサーチを行なう場合には、次のSSCH実行前に別命令でポインタの更新が必要になる。SMOV, SCMPでも同様である。

【0471】[ポインタの更新後に命令を終了するデメリット] 命令実行後のポインタ値が終了条件（サーチ条件）成立のエレメントよりも進んでいるので、SSCH命令としてはすなおな仕様ではない。BVSCCH, BSCH命令の仕様とも合わない。TRONCHIPでは、[ポインタの更新前に命令を終了するデメリット]の方
20 を重視し、終了条件成立の場合には、ポインタの更新後*

BSCH/F 現在ポインタの指しているものからサーチする。

サーチ終了後のポインタは、見付かったデータを指す。

SSCH/F 現在ポインタの指しているものからサーチする。

サーチ終了後のポインタは、見付かったデータの次を指す。

QSCH/F 現在ポインタの指している次のものからサーチする。

サーチ終了後のポインタは、見付かったデータを指す。

ストリング命令の場合、エレメント数R2は符号なしの数として扱われる。これは、R2を符号なしと考えることにより、R2=0の指定によってエレメント数をH'10000000と解釈し、エレメント数による終了を行なわないようにできるためである。この機能は、言語Cのstrcmp関数の実現などに利用できる。また、インプリメント上も、R2を符号なしと考える方がエレメント数による終了の判定が楽になる。一方、ビットフィールド命令のwidthは、以下のような理由により、固定長ビットフィールド命令、任意長ビットフィールド命令とも符号付きとして扱うことにする。

【0473】・命令実行時、ビットフィールド命令のwidthはoffsetに加算される形になるが、offsetは符号付きである。widthを符号なしとすると、符号付きと符号なしの数を足すことになり、すっきりしない。ストリング命令のエレメントサイズの場合は、サイズを乗じた上でポインタに加算するのであるから、符号なしの方が自然である。

・命令実行の上で符号付きか符号なしかの違いが出てくるのは、任意長ビットフィールド命令でwidthがH'80000000~H'ffffffffの場合である。widthを符号付きとしておけば、width
50

332

*に命令を終了するという仕様にする。したがって、SMOV/F, SCMP/F, SSCH/F, /R命令終了後のポインタは、終了条件の成立したエレメントの次のエレメントを指すことになる。また、SMOV/B, SCMP/B命令の場合はプリデクリメントでポインタを更新するため、命令終了後のポインタは終了条件の成立したエレメントを指すことになる。SMOV/B, SCMP/Bとの仕様を合わせるという意味で、BVMA P/B, BVCPY/Bの場合にも、演算の対象となるビットフィールドの最大オフセット+1をR1, R4で指定する。

【0472】ただし、BVSCCH, BSCHについては、命令終了後のビットオフセットが直接サーチ対象のビットを指している方が便利だと考えられるため、/F, /Bともそのような仕様にする。また、QSCHについては、ポインタがプリ更新となっているため、SSCH, BSCHとはポインタ更新のタイミングが異なっている。結局、BSCH/F (BVSCCH/F), SSCH/F, QSCH/Fのサーチのパターンをまとめると、次のようになる。

がこの値の時V_flagをセットするだけで命令を終了するが、widthを符号なしとした場合には、widthがこの値の時もビットフィールド操作を行なうことになる。しかし、widthがH'80000000~H'ffffffffだとすると、offset+widthの値を符号付きとして扱う場合には既にオーバーフローしているし、offset+widthを符号なし（あるいは33ビット符号付き）として扱う場合にも、offsetの値によってはオーバーフローする。オーバーフローに関しては、「offset+widthがオーバーフローした場合には動作を保証しない」となっているのであるから、同じようなインプリメントをするのであれば、widthを符号なしとしても「動作を保証しない」ケースが増えるだけである。widthを符号なしとして、しかもwidth>H'80000000の場合の動作を保証するのであれば、ハードウェアの負担を伴う。

・ストリング命令の場合は、終了条件によって命令を終了する場合があったため、エレメントサイズによる終了をしたくない時に0を設定するという使い方がある。0で無限大（H'100000000）を表現するには、どうしてもエレメントサイズを符号なしとして扱う必要

333

があった。これに対して、BVMAP, BVCPYではwidth以外に命令終了の要素がないため、プログラミング上、widthとして必ず意味のある値を設定することになる。その場合は、「レジスタ上の値は原則として符号付きの数と考える」という原則に合わせる方が自然である。

【0474】 [ストリング命令、任意長ビットフィールド命令の基本原則のまとめ]

・サーチ系の命令では、ポインタ更新のタイミングはサーチ方向に依存しない。BSCH, BVSCHでは、/F, /Bともサーチ終了後のポインタは見付かったビットを指す。SSCHでは、/F, /Rともサーチ終了後のポインタは見付かったエレメントの次のエレメントを指す。

・実際にデータを操作する命令では、/Fの場合にポストインクリメント、/Bの場合にプリデクリメントの形で処理を行なう。これに該当するのは、SMOV, SCMP, BVMAP, BVCPYである。SSTR, BV PATは/Fのみであるが、やはり同じ原則にあてはま*

0 ==> V_flag

repeat

R2 - .1 ==> R2

mem[R0] ==> mem[R1] ==> temp

R0 + size ==> R0

R1 + size ==> R1

compare temp with R3, R4 and set F_flag, M_flag

according to eeee

/* 終了条件成立の場合 F_flag=1となる */

if (F_flag = 1) then exit

check_interrupt

until (R2 = 0)

1 ==> V_flag

334

＊る。

・ストリング命令では、エレメントサイズは符号なしとして扱われ、' 0 ' の場合はH' 1 0 0 0 0 0 0 0 0 を表わす。一方、任意長ビットフィールド命令では、widthは符号付きとして扱われ、widthがH' 0 0 0 0 0 0 1 ~ H' 7 f f f f f f f f の場合にのみ実際のビットフィールド操作を行なう。

【0475】 A11-2. ストリング命令の詳細仕様 SMOV

10 SMOVのオペレーションをまとめると、次のようになる。ただし、最終的な結果が同じであれば、メモリアクセスの順番が下記のものとは異なっても構わない（他の高機能命令も同様）。また、srcとdestがオーバーラップしていた時に、正しくない方のオプションを使用した場合 (src < destで/Fを使用した場合、およびsrc > destで/Bを使用した場合) の動作も、下記の通りでなくてもよい。

【0476】 [SMOV/Fのオペレーション]

【0477】 [SMOV/Bのオペレーション]

```

335
0 ==> V_flag
repeat
    R2 - 1 ==> R2
    R0 - size ==> R0
    R1 - size ==> R1
    mem[R0] ==> mem[R1] ==> temp
    compare temp with R3, R4 and set F_flag, M_flag
    according to eeee
    /* 終了条件成立の場合 F_flag=1となる */
    if ( F_flag = 1) then exit
    check_interrupt
until (R2 = 0)
1 ==> V_flag

```

336

SMOVでは、R2の初期値が何であっても、かならず1個以上のエレメントは処理される。SMOVの終了要因をまとめると、次のようになる。

1. エレメント（データ）数（R2）による終了
エレメント数によって命令を終了した場合には、V_flag = 1となる。2のケースとは同時には起こらない。

2. 終了条件による終了
この時は、F_flag = 1となる。終了条件を満足したエレメントの転送も行なわれる。

【0478】SCMP

SCMPでは、エレメント数による命令の終了と終了条件による命令の終了のほかに、比較データの不一致によって命令を終了することがある。SCMPでデータの不一致により命令を終了した場合にも、終了条件により命

令を終了した場合と同様に、ポインタの更新が行なわれてから命令が終了する。SCMPでは、エレメント数による終了要因と他の終了要因とが同時に発生することはないが、終了条件とデータ不一致の終了要因が同時に満たされる可能性はある。SCMPがエレメント数によって終了した場合には、次のエレメントの比較は行なわず、次のエレメントが不一致または終了条件成立であっても、V_flag = 0, F_flag = 0, Z_flag = 1として命令を終了する。SCMPのオペレーションをまとめると、次のようになる。ただし、最終的な結果が同じであれば、メモリアクセスの順番が下記のものとなっても構わない。これと等価の動作をすればよい。

30 【0479】[SCMP/Fのオペレーション]

```

337
0 ==> V_flag
repeat
    R2 - 1 ==> R2
    mem[R0] ==> temp1
    mem[R1] ==> temp2
    R0 + size ==> R0
    R1 + size ==> R1
    compare temp1 with temp2 and set Z_flag,
    L_flag, X_flag
    /* データ不一致の場合 Z_flag=0となる */
    compare temp1 with R3, R4 and set F_flag, M_flag
    according to eeee
    /* 終了条件成立の場合 F_flag=1となる */
    if ( F_flag = 1 .or. Z_flag = 0) then exit
    /* 終了条件またはデータ不一致による終了
*/      check_interrupt

```

```

until (R2 = 0)
1 ==> V_flag

```

* 【0480】 [SCMP/Bのオペレーション]

```

*
0 ==> V_flag
repeat
    R2 - 1 ==> R2
    R0 - size ==> R0
    R1 - size ==> R1
    mem[R0] ==> temp1
    mem[R1] ==> temp2
    compare temp1 with temp2 and set Z_flag,
    L_flag, X_flag
    /* データ不一致の場合 Z_flag=0となる */
    compare temp1 with R3, R4 and set F_flag, M_flag
    according to eeee
    /* 終了条件成立の場合 F_flag=1となる */
    if ( F_flag = 1 .or. Z_flag = 0) then exit
    /* 終了条件またはデータ不一致による終了
*/      check_interrupt
until (R2 = 0)
1 ==> V_flag

```

【0481】 SCMPの終了要因をまとめると、次のようになる。

1. エレメント (データ) 数 (R2) による終了
この時、Z_flag=1, F_flag=0, V_flag=1となる。2, 3のケースとは同時には起こら

ない。

2. 終了条件による終了

この時は、F_flag=1となる。終了条件を満足したエレメントの比較も行なわれ、その比較結果がZ_flag, L_flag, X_flagに設定される。比

339

較が不一致だった場合は、2, 3の2つの終了要因が同時に満たされたことに相当する。

3. 比較中のエレメントの不一致による終了

この時は、不一致のあったエレメントの比較結果がZ__flag (=0), L__flag, X__flagに設定される。V__flagは0となる。

【0482】SSCH

```
0 ==> V__flag
repeat
    R2 - 1 ==> R2
    mem[R0] ==> temp
    R0 + size ==> R0
    compare temp with R3, R4 and set P__flag, M__flag
    according to eeee
    /* 終了条件成立の場合 F__flag=1となる */
    if ( F__flag = 1) then exit
    /* 終了条件 (検索条件) による終了 */
    check_interrupt
until (R2 = 0)
1 ==> V__flag
```

【0483】[SSCH/Rのオペレーション]

```
0 ==> V__flag
repeat
    R2 - 1 ==> R2
    mem[R0] ==> temp
    R0 + R5 ==> R0
    compare temp with R3, R4 and set F__flag, M__flag
    according to eeee
    /* 終了条件成立の場合 F__flag=1となる */
    if ( F__flag = 1) then exit
    /* 終了条件 (検索条件) による終了 */
    check_interrupt
until (R2 = 0)
1 ==> V__flag
```

SSCHの終了要因をまとめると、次のようになる。

1. エレメント (データ) 数 (R2) による終了
この時、V__flag=1となる。2のケースとは同時には起こらない。

2. 終了条件 (検索条件) による終了
この時は、F__flag=1となる。

【0484】SSTR

SSTRではフラグ変化は起こらない。SSTRのオペレーションをまとめると次のようになる。

[SSTRのオペレーション]

340

*SSCHが終了条件 (検索条件) によって終了した場合には、/F, /Rとも、命令実行後のポインタは終了条件の成立したエレメントの次のエレメントを指す。また、SSCHがエレメント数によって終了した場合にも、命令実行後のポインタは次のエレメントを指す。SSCHのオペレーションをまとめると次のようになる。

* [SSCH/Fのオペレーション]

```
repeat
    R2 - 1 ==> R2
    R3 ==> mem[R1]
    R1 + size ==> R1
    check_interrupt
until (R2 = 0)
```

【0485】A11-3. 高機能命令終了時のレジスタ値

50 TRONCHIPで高機能命令を実行した場合、命令終

341

342

了時の各レジスタの値は次のようになる。なお、 RXi $RXend$ は命令実行後のレジスタRXの値を示す。
 $n i t$ は命令実行前のレジスタRXの値を示す。また、
 [BVSCH]

/Fの時は、オフセットが $Rlinit \sim Rlinit + R2init - 1$ の範囲をサーチする。

/Bの時は、オフセットが $Rlinit \sim Rlinit - R2init + 1$ の範囲をサーチする。

$R2init(width) \leq 0$ の場合は、 V_flag をセットして命令を終了する。

$R1, R2$ は不変である。

- サーチが成功した場合

$RO(base\ address)$: 不変

$R1(offset)$: サーチ結果

見付かったビットのビットオフセット。

$R2(width)$: まだサーチしていないビットフィールドと見付かったビットとを合わせたビットフィールドの長さ
 つまり、/Fであれば $R2init + Rlinit - Rlend$.
 /Bであれば $R2init - Rlinit + Rlend$ となる。

- サーチが失敗した場合

$RO(base\ address)$: 不変

$R1(offset)$: 最後にサーチしたビットの次のビットオフセット。
 つまり、/Fであれば $Rlinit + R2init$.
 /Bであれば $Rlinit - R2init$ となる。
 これはBSCHと同じ考え方である。

$R2(width)$: 0

[0486]

343

344

[BVMAP]

[BVCPY]

/Fの時は、R1init~R1init+R2init-1 のビットオフセットを持つ領域がsrc、R4init ~R4init+R2init-1 のビットオフセットを持つ領域がdestとなる。

/Bの時は、R1init~R1init-R2init+1 のビットオフセットを持つ領域がsrc、R4init-1~R4init-R2init のビットオフセットを持つ領域がdestとなる。

R2init(width) ≤ 0 の場合は、何もせずに命令を終了する。

R1, R2, R4 は不変である。

R0(src base): 不変

R1(src offset): /FであればR1init+ R1init, /B であればR1init-R2init となる。

R2(width): 0

R3(dest base) 不変

R4(dest offset) /FであればR4init+R2init, /BであればR4init-R2init となる。

R5 (演算の種類) 不変 (BVMAP のみ)

【0487】

[BVPAT]

R4init ~R4init+R2init-1 のビットオフセットを持つ領域がdestとなる。

R2init(width) ≤ 0 の場合は、何もせずに命令を終了する。

R2, R4 は不変である。

R0(pattern): 不変

R2(width): 0

R3(dest base): 不変

R4(dest offset): R4init+R2init

R5 (演算の種類) 不変

【0488】 [SMOV]

345

346

/F の時は、

$R0init \sim R0init + R2init * \text{エレメントサイズ} - 1$ のアドレスを持つ領域がsrc、

$R1init \sim R1init + R2init * \text{エレメントサイズ} - 1$ のアドレスを持つ領域がdestとなる。

/B の時は、

$R0init-1 \sim R0init-R2init$

*エレメントサイズのアドレスを持つ領域がsrc、

$R1init-1 \sim R1init-R2init$

*エレメントサイズのアドレスを持つ領域がdestとなる。

例えば、H' 0000～H' 00ffのストリングをH' 0300～H' 03ffに転送する場合、SMOV/F.Wでコピーすると、

$R0 = H' 0000, R1 = H' 0300, R2 = H' 0040$

SMOV/B.Wでコピーすると、

$R0 = H' 0100, R1 = H' 0400, R2 = H' 0040$

とすればよい。ただし、終了条件が成立した場合には、もdest側に転送される。処理が途中で打ち切られる。終了条件の成立したデータ 20 【0489】—エレメント数によって終了した場合 (V_flag=1)

R0(src address): /Fであれば $R0init + R2init * \text{エレメントサイズ}$ 、

/Bであれば $R0init - R2init * \text{エレメントサイズ}$

R1(dest address): /Fであれば $R1init + R2init * \text{エレメントサイズ}$ 、

/Bであれば $R1init - R2init * \text{エレメントサイズ}$

R2(エレメント数): 0

R3(終了条件1): 不変

R4(終了条件2): 不変

- 終了条件が成立して終了した場合(F_flag=1)

R0(src address): /Fの時終了条件の成立したsrcのエレメントの次のエレメントのアドレス

/Bの時終了条件の成立したsrcのエレメントのアドレス

R1(dest address): /Fの時終了条件の成立したsrcのエレメントの次のエレメントを転送すべきdestのアドレス

/Bの時終了条件の成立したsrcのエレメントを転送すべきdestのアドレス

/F./Bとも $R1init + R0end - R0init$ となる。

R2(エレメント数): まだ転送されていないエレメントの数

/Fの時 $R2init - (R0end - R0init) / \text{エレメントサイズ}$ 、

/Bの時 $R2init - (R0init - R0end) / \text{エレメントサイズ}$ となる。

R3(終了条件1): 不変

R4(終了条件2): 不変

【0490】 [SCMP]

347

348

/F の時は、

R0init ~ R0init+R2init*エレメントサイズ-1のアドレスを持つ領域がsrc1、R1init ~ R1init+R2init*エレメントサイズ-1のアドレスを持つ領域がsrc2となる。

/B の時は、

R0init-1 ~ R0init-R2init * エレメントサイズのアドレスを持つ領域がsrc1、R1init-1 ~ R1init-R2init * エレメントサイズのアドレスを持つ領域がsrc2となる。

例えば、H' 0000 ~ H' 00ffのストリングをH' 0300 ~ H' 03ffのストリングとを比較する場合、

SCMP/F.W で比較すると、

R0=H' 0000, R1=H' 0300, R2=H' 0040

SCMP/B.W で比較すると、

R0=H' 0100, R1=H' 0400, R2=H' 0040

*ントについても比較が行なわれ、その結果がL__flag, X__flag, Z__flagにセットされる。また、比較中に不一致のエレメントが見付かった場合にも、処理が途中で打ち切られる。

とすればよい。ただし、終了条件が成立した場合には、

処理が途中で打ち切られる。終了条件の成立したエレメ * 20 【0491】—エレメント数によって終了した場合 (V__flag=1)

R0(src1 address): /FであればR0init+R2init*エレメントサイズ、

/BであればR0init-R2init*エレメントサイズ

ただし、R2init<0 の場合には無変化

R1(src2 address): /FであればR1init+R2init*エレメントサイズ、

/BであればR1init-R2init*エレメントサイズ

R2(エレメント数): 0

R3(終了条件1): 不変

R4(終了条件2): 不変

—終了条件の成立、またはエレメント値の不一致によって終了した場合

(F__flag=1.or. Z__flag=0)

R0(src1 address): /Fの時終了条件の成立した(または不一致であった)src1のエレメントの次のエレメントのアドレス

/Bの時終了条件の成立した(または不一致であった)src1のエレメントのアドレス

R1(src2 address): /Fの時終了条件の成立した(または不一致であった)src1のエレメントの次のエレメントと対応するsrc2のエレメントのアドレス

/Bの時終了条件の成立した(または不一致であった)src1のエレメントと対応するsrc2のエレメントのアドレス

349

350

/F./BともRlinit+ROend-ROinitとなる。

R2(エレメント数): まだ比較されていないエレメントの数

/Fの時R2init-(ROend-ROinit)/エレメントサイズ、

/Bの時R2init-(ROinit-ROend)/エレメントサイズ

となる。

R3(終了条件1): 不変

R4(終了条件2): 不変

【0492】[SSCH]

10

/Fの時は、

ROinit ~ ROinit+R2init*エレメントサイズ-1のアドレスを持つ領域を検索する。

/Rの時は、

ROinit ~ ROinit+R5*R2init-1のアドレスを持つ領域を、R5毎に飛び飛びに検索する。

ただし、終了条件(検索条件)が成立した場合には、処理が途中で打ち切られる。

—エレメント数によって終了した場合 (V__flag = 1)

R0(src address): /FであればROinit+R2init*エレメントサイズ、

/RであればROinit+R2init*R5

R2(エレメント数): 0

R3(終了条件1): 不変

R4(終了条件2): 不変

R5(カウンタ 更新値): 不変

—終了条件(検索条件)が成立して終了した場合 (F__flag = 1)

R0(src address): 終了条件の成立したsrc のエレメントのアドレス

R2(エレメント数): 終了条件の成立したエレメントと、まだ検索されていないエレメントの合計数

/Fの時R2init-(ROend-ROinit)/エレメントサイズ、

/Rの時R2init-(ROend-ROinit)/R5となる。

R3(終了条件1): 不変

R4(終了条件2): 不変

R5(カウンタ 更新値): 不変

【0493】[SSTR] Rlinit~Rlinit+R2init*エレメントサイズ-1のアドレスを持つ領域に、R3で指定されるデータを繰り返し書き込む。他のストリング命令とは異なり、終了条件の指定は行なわない。また、フラグのセットも行なわない。R2init(width) ≤ 0の場合は、即座に命令を終了する。R1, R2は不変である。

R1(dest address): Rlinit+R2init*エレメントサイズ

R2(エレメント数): 0

R3(書き込みデータ): 不変

【0494】[QSCH]

—キュー終了値(R2)によって終了した場合 (V__flag = 1)

351

352

R0(entry address):R2init

R1(previous entry):R0endで示されるエントリの直前 (/Fの場合) または直後
(/B の場合) のエントリのアドレス

R2(キュー終了値): 不変

R3(終了条件1): 不変

R4(終了条件2): 不変

R5(オフセット): 不変

R6(マスク): 不変

—終了条件(検索条件)が成立して終了した場合 (F__ flag=1)

R0(entry address):終了条件の成立しているキューエントリのアドレス

R1(previous entry):R0endで示されるエントリの直前 (/Fの場合) または直後

(/Bの場合) のエントリのアドレス

R2(キュー終了値): 不変

R3(終了条件1): 不変

R4(終了条件2): 不変

R5(オフセット): 不変

R6(マスク): 不変

【0495】付録12. オペランドが干渉した場合の動作

一つの命令が複数のオペランドを持ち、その中で、@SP+, @-SPモードとSPを参照するモードとを併用した場合には、@SP+, @-SPモードによって更新されたSP値がいつから反映されるかということが問題になる。この問題をより一般的に考えると、オペランドが干渉していた場合の動作を明確に規定すればよいということになる。この資料は、本発明装置で、命令中のオペランドが干渉を起こした場合に、その動作をきちんと定義することを目的としたものである。資料の最初の方

でこの問題に対する考え方を述べ、資料の後半では、実際に本発明装置の動作の規定を行っている。

【0496】A12-1. 命令実行モデル

オペランド干渉の問題を整理するために、まず、各命令からメモリやレジスタに対して行われるread, write, read-modify-writeの基本アクセス操作をすべて含むような仮想的な命令パターンを考え、それを以下のようにモデル化する。

RA1. 第一readオペランド(R01) 実効アドレス計算

↓

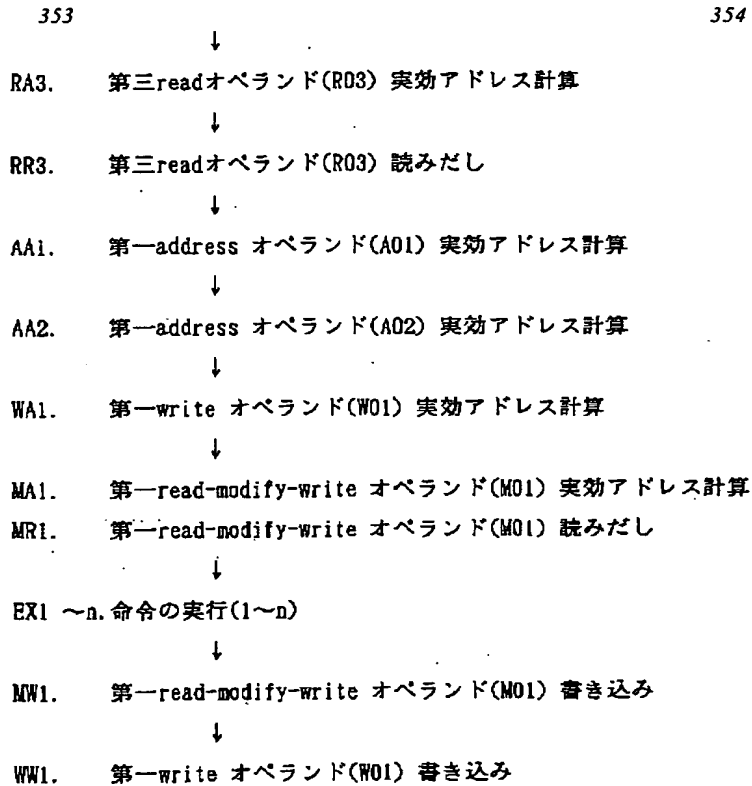
RR1. 第一readオペランド(R01) 読みだし

↓

RA2. 第二readオペランド(R02) 実効アドレス計算

↓

RR2. 第二readオペランド(R02) 読みだし



【0497】・モデルを構成するRA1~WW1の夫々の要素を「ステップ」と呼ぶ。

・@SP+, @-SPのアドレッシングモードを指定した場合、SP値の更新は「実効アドレス計算」のステップ(RA1, RA2, RA3, MA1, WA1)で行われるものとする。

・各ステップは、次のいずれかのパターンでリソースをアクセスする。ただし、「リソース」とはメモリまたはプログラムから見えるレジスタを指す。

ーリソースアクセスなし

ー一つまたは複数のリソースからのread

ー一つのリソースへのwrite (EX1~EXn, MW1, WW1のみ)

ーリソースSPのread-modify-write (RA1, RA2, RA3, MA1, WA1, WA2)

・各ステップでリソースの値の更新があった場合には、次のステップから更新された値が反映される。

・一般の命令では、このうち一部のステップのみが存在する。

・EX1~EXnのステップは、実際には、命令の種類によってEX1, EX2, EX3...のいくつかのステップに分かれている。

上記のモデルでは、各ステップで書き込みを行なうリソースが高々一つであること、SPのread-modify-writeのケースを除けば、一つのステップ内でreadとwriteが混在するケースはないこと、リソースに書き込んだ値が次のステップから反映される

こと、といった条件を明らかにしているので、ステップ間で共通のオペランドをアクセスした場合にも動作は明確に規定される。したがって、各命令の動作を上記のモデルにうまく当てはめることができれば、オペランドが重なっていた場合の動作もきちんと規定することができる。本発明装置のほとんどの命令の動作は、上記の実行パターンのサブセットとして成り立っている。ただしどの動作がどのステップに対応するかということについては、命令動作の意味付けによっていろいろな解釈ができる。例えばACB:G @SP+, SP, @SP+, newpcの場合には、次の①のような解釈をするのが自然である。つまり、ACB命令の各動作ステップやオペランドと、上記のモデルとの対応は、次の①のように行なうのが普通であり、実際、これが正しいTRON仕様となっている。

【0498】①ACBのSTEPオペランドがモデルのR01に対応

ACBのLIMITオペランドがモデルのR02に対応
xregオペランドはEX1~EX3の中だけで扱い、R0n, M0n, W0nとは考えない。

RA1. SP==>roladdr; SP+size==>SP

RR1. @roladdr==>step

RA2. SP==>ro2addr; SP+size==>SP

RR2. @ro2addr==>limit

RA3 ~MR1. なし

355
 EX1. SP==>xreg
 EX2. xreg+step==>xreg
 if(xreg<limit) jump to newpc endif
 EX3. xreg==>SP
 MW1 ~WW1. なし

356
 *—roladdr, ro2addr, step, limit
 it xregは内部変数であり、モデルで使用する意味での「リソース」には含めないこの場合、命令実行前のSP値をinitSPとすると、

*

step: @(initSP)
 xreg初期値: initSP+size*2
 limit: @(initSP+size)
 EXの後のxreg: initSP+size*2+@(initSP)
 ジャンプ条件: initSP+size*2+@(initSP) <@(initSP+size)
 SP最終値: initSP+size*2+@(initSP)

しかし、実際のACB命令の動作とモデルとの対応を次の②のようにすることも不可能ではない。②の動作は、厳密には①と同じではない。

【0499】②ACBのstepオペランドがモデルのR01に対応
 ACBのxregオペランドが読みだし時R02に、書き込み時W01に対応ACBのlimitオペランドがモデルのR03に対応

RA1. SP==>roladdr; SP+size==>SP
 RR1. @roladdr==>step
 RA2. 何も参照しない
 RR2. SP==>xreg
 BA3. SP==>ro3addr; SP+size==>SP
 RR3. @ro3addr==>limit

AA1 ~WR1. なし

EX. xreg+step==>xreg
 if(xreg<limit) jump to newpc endif

※

step: @(initSP)
 xreg初期値: initSP+size
 limit: @(initSP+size)
 EXの後のxreg: initSP+size+@(initSP)
 ジャンプ条件: initSP+size+@(initSP) <@(initSP+size)
 SP最終値: initSP+size+@(initSP)

これは①とは異なる動作である。仕様書の命令の解説の項だけでは①が正しいか②が正しいかは厳密にはわからないため、命令によっては、モデルとの対応付けの曖昧な場合が出てくる。モデルとの対応付けが違っていると、上の例のように細かい動作が異なってくることがある。この資料では本発明装置の命令について、上記のような方法で命令動作とモデルとの対応を決め手いくことにより、オペランドが重なった場合の動作を明確に規定することを目的としている。

【0500】A12-2. 基本原則

命令について個別に検討する前に、オペランドの干渉に

※ MW1. なし

WW1. xreg==>SP

—roladdr, ro3addr, step limit
 it, xregは内部変数であり、モデルで使用する意味での「リソース」には含めないこの場合、命令実行前のSP値をinitSPとすると、

40 ついての基本原則を述べる。

(原則1) 短縮形と一般形では、全く同じ動作をしなければならない。すなわち、命令フォーマットの違いはオペランド干渉の問題には影響しない。短縮形は完全に一般形のサブセットとなるべきであり、動作の異なる短縮形は混乱を招く。命令実行モデルは、インプリメントへの依存性やインプリメントの都合を考えたものではないため、場合によっては、規定した通りにインプリメントできないことがある。例えば、ADD: L @SP+, SPがADD: G @SP+, SPと異なった動作をする場合があるというのは、この例である。このようなケ

357

ースに関しては、命令実行モデルの方はそのままにして
おき、場合に応じて例外として扱う。

(原則2) 基本的には、命令ビットパターンの中でのオペランドの出現順序にしたがって、命令のオペランドとモデルでの第一readオペランド(R01)、第二readオペランド(R02)・・・との対応を決める。こうしておけば、一般にはインプリメント方法とも矛盾しない。命令実行モデルは、各命令の動作とビットパターンから考えて、最も自然な形になるようにする。短縮形と一般形を持つ命令の場合、命令実行モデルと実際の命令動作との対応が自然になるかどうかは、どのフォーマットの命令ビットパターンを基準にするかによって異なるが、この場合は減速として一般形を基準にする。命令動作とモデルとの対応という点に関しては、命令ビットパターンの都合は考えるが、インプリメントの都合は特に考えない。

(原則3) 原則2があっても、汎用アドレッシングでないオペランド(ACBのxregなど)については、モデルでの第n-readオペランド(R0n)、第m-writeオペランド(W0m)としては扱わず、EXの

ステップ中だけで扱うのが普通である。
(原則4) また、命令の意味の中に暗黙に含まれているメモリアクセス(PUSH, POPでのスタック操作など)についても、モデルでの第n-readオペランド(R0n)、第m-writeオペランド(W0m)としては扱わず、EXのステップ中だけで扱うのが普通である。

さらに、各命令ごとに特殊な事情のある場合や、多数のオペランドを持つ場合(LDM, STMなど)には、これらの原則が当てはまらないことがある。詳しくは命令ごとに検討する。

【0501】A12-3. 各命令での実際

本発明装置の命令をいくつかのパターンに分類してモデルとの対応付けを行ない、オペランドが干渉した場合の動作を明確にする。(原則1)により、命令フォーマットが違ってもオペランドが干渉した場合の動作は変わらないので、命令フォーマットの違いによる場合分けを行なう必要はない。なお、動作例の中に出てくるinit SPは、命令実行前のSPの意味である。また、動作例で出てくる命令のオペランドサイズは、断わりのない限りすべて32ビットとする。'==>'は値の代入を、'【〜】'はコメントを示す。RA1~WW1のステップのうち、その命令で使用していないステップについては、説明を省略してある。

【0502】Oオペランド(N)

```
PUSH    src
LDPSB   src
LDPSM   src
JRNG    vector
```

358

該当命令：

```
NOP
PIB
RTS
RRNG
TRAP
REIT
STCTX
PTLB
BVSCH
BVMAP
BVCPV
BVPAT
SMOV
SCMP
SSCH
SSTR
QSCH
```

命令の動作は、すべてモデルのEXのステップだけで行なわれ、それ以外の部分は存在しない。したがって、オペランドの干渉については特に問題とはならない。高機能命令では、EXのステップ数が有限とはならない場合もある。しかし、これはオペランドの干渉の問題には直接関係しないので、ここでは議論しない。

[命令実行モデルとオペランドとの対応]

EX. 命令固有の動作

【0503】1オペランドイミディエート(I)

当該命令：

```
BRA      newpc
Bcc       newpc
BSR       newpc
TRAPA     vector
WAIT      imask
```

オペランドを一つもつが、オペランドの値は命令コード中で直接指定されるため、オペランドの干渉については問題とならない。newpc, vector, imaskはEXのステップで操作すると考える。

[命令実行モデルとオペランドとの対応]

EX. 命令固有の動作(newpc, vector, imaskの参照を含む)

【0504】1オペランドread(R)

該当命令：

359

360

このパターンの命令の場合、src, vectorをモデルの第一readオペランド(R01)として扱う。

*では、既にSPのインクリメントが行なわれていることになる。

src=@SP+の場合、命令固有の動作を行なう時点*

[命令実行モデルとオペランドとの対応]

RA1. src, vector のアドレス計算==>roladdr

RR1. @roladdr==>src, vector

EX. 命令固有の動作(命令による暗黙のスタック操作を含む)

PUSHでは src==>PSB

LDPSB では .op.(PSB, src)==>PSB

LDPSM では .op.(PSM, src)==>PSM

例えばPUSH@SP+にこのモデルを適用すると、EXでPUSH命令固有の動作(src==>@-SP)を行なう前に、RA1.で既にsrc=@SP+によるSPの更新が行われることがわかる。ただし、間違いを防ぐため、PUSH@SP+は禁止(RIE)となっている(図396参照)。

※srcaddr, prgaddr, ctxaddr, chkaddr, newpcをモデルの第一addressオペランド(A01)として扱う。なお、AA1では@SP+, @-SPのモードは利用できない。
[命令実行モデルとオペランドとの対応]

【0505】1オペランドaddress(A)

該当命令:

PUSHA srcaddr

PSTLB prgaddr

20

LDCTX ctxaddr

ACS chkaddr

JMP newpc

JSR newpc

※

AA1. srcaddr, prgaddr, ctxaddr, chkaddr, newpc のアドレス計算
==>aoladdr

EX. 命令固有の動作(命令による暗黙のスタック操作を含む)

PUSHA では aoladdr==>@-SP

JSR では PC==> @-SP: aoladdr ==>PC

PUSHA @SPとJSR @SPの動作は図397のとおり。

【0506】1オペランドwrite(W)

該当命令:

POP dest

STPSB dest

STPSM dest

★40

WA1. destの実効アドレス計算==>woladdr

EX. 命令固有の動作(命令による暗黙のスタック操作を含む)

POP では @SP+==> dest2

STPSB では .op.(PSB)==>dest2

STPSM では .op.(PSM)==>dest2

WW1. dest2==>@woladdr

例えばPOP @-SPにこのモデルを適用すると、EX.でPOP命令固有の動作(@SP+==> dest2)を行なう前に、WA1.で既にdest=@-SP

★destを第一writeオペランド(W01)として扱う。この場合、EXのステップより前のWA1のステップでdestの実効アドレス計算が行なわれ、値の書き込みのみEXより後のWW1のステップで行なわれることになる。

[命令実行モデルとオペランドとの対応]

によるSPの更新が行なわれることがわかる。また、POP @ (d, SP)にこのモデルを適用すると、EX.でPOP命令固有の動作(@SP+==> dest

361

2)を行なう前に、WA1.でdest=@(d, SP)のアドレス計算が行なわれ、destのアドレス計算にはinitSPが使用されることがわかる。ただし、間違いを防ぐため、実際にはPOP @-SPは禁止(RIE)となっている。また、POP @(d, SP)については、@(d, Rn)のアドレッシングモードでRn=SPの時のみ実行禁止とするのは無理があること、スタック操作においてPOP @ (disp, SP)を使用する用途も考えられること、により、禁止にはしていない。POP命令の動作例は図398のようになる。

【0507】1オペランドrmw (M)

該当命令:

```
NEG    dest
NOT     dest
SHXL    dest
SHXR    dest
```

destをM01に対応させる。この場合、MA1で@SP+, @-SPを指定することはできない。

〔命令実行モデルとオペランドとの対応〕

```
MA1.    destの実効アドレス計算==>moladdr
MR1.    @moladdr==> dest1
EX.      命令固有の動作
        .op.(dest1) ==> dest2
MW1.    dest2 ==> @moladdr
```

【0508】2オペランドread~write (RW)

該当命令:

```
MOV     src, dest
MOVU    src, dest
RVBY    src, dest
RVBI    src, dest
PACKss  src, dest
UNPKss  src, dest, adj
```

srcをR01、destをW01に対応させる。した*

```
AA1.    src, srcaddr, queue の実効アドレス計算==> aoladdr
WA1.    destの実効アドレス計算==> woladdr
```

EX. 命令固有の動作

```
.op.(aoladdr) ==> dest2
```

```
MW1.    dest2 ==> @woladdr
```

STC, MOVAの動作例との対応は図401のとおり。mov @SP, @-SPの場合、srcaddrの実効アドレス計算(AA1)のステップで既にSPが参照されており、destの実効アドレス計算(WA1)ステップにおけるSPの更新はsrcaddrに反

362

*がって、src側の実効アドレス計算とそれに伴うSPの更新がすべて終わってからsrcのフェッチを行ない、その後dest側の実効アドレス計算を行ない、最後に命令固有の動作を行って結果をdestにセットする。UNPKssのadjはEXのステップで扱い、n-readオペランドとはしない。

〔命令実行モデルとオペランドとの対応〕

```
RA1.    srcの実効アドレス計算==> roladdr
RR1.    @roladdr==> src1
WA1.    destの実効アドレス計算==> woladdr
EX.      命令固有の動作
        .op.(src1) ==> dest2
        UNPKssではadjを参照
MW1.    dest2 ==> @woladdr
```

MOVの動作例は図399のとおり。PACK, UNPKの動作例は図400のとおり。

【0509】2オペランドaddress~write (AW)

該当命令:

```
STC     src, dest
STP     src, dest
MOVA    srcaddr, dest
MOVPA   srcaddr, dest
STATE   srcaddr, dest
QDEL    queue, dest
```

制御空間、物理空間など特殊な空間からの読みだしを行なう命令では、特殊空間側の実効アドレスsrc, srcaddrをA01として扱い、特殊空間の実際のアクセスはEXのステップで行なう。また、QDEL命令でも、キューエントリを指定する側の実効アドレスqueueをA01として扱い、キューリンクの実際の操作はEXのステップで行なう。こうしておけば、命令ビットパターンやインプリメント方法ともうまく対応する。AA1では、@SP+, @-SPを指定することはできない。destはW01として扱う。

映されない。これに対して、STC @sp0, @-SPがモデル通りの動作をするものと考え、srcの実効アドレス計算(AA1)のステップではまだSPが参照されておらず、制御レジスタとしてのSPのアドレス(H'0124)が計算されるだけである。SPが参照されるのはEXのステップであり、これはdestの実効アドレス計算(WA1)ステップにおいてSPを更新した後である。したがって、initSP-4がdestに転送されることになる。ただし、STC @sp

363

0, @-SPの動作については、以上のようなモデル通りの動作をするかどうかインプリメント依存になっている。つまり、モデル通り `init SP-4` を転送する場合と、モデルに対しては例外の扱いとなるが、`init SP` を転送する場合とがある。

【0510】2オペランド `read~address` (RA)

該当命令:

```
LDC    src, dest
LDP    src, dest
LDATE  src, destaddr,
BTST   offset, base
BSET   offset, base
BCLR   offset, base
BNOT   offset, base
BSETI  offset, base
BCLRI  offset, base
BFEXT  offset, width, base, dest
BFEXTU offset, width, base, dest
BFINS  src, offset, width, base
BFINSU src, offset, width, base
BFCMP  src, offset, width, base
BFCMPU src, offset, width, base
CHK    bound, index, xreg
```

【0511】制御空間、物理空間など特殊な空間への書き込みを行なう命令 `LDC`, `LDP`, `LDATE` では、特殊空間側の実効アドレス `dest`, `destaddr` を `A01`, `src` を `R01` として扱う。特殊空間の実際のアクセスは `EX` で行なう。ビット操作命令では、`ba`

`RAI`. `src, offset, index` の実効アドレス計算 ==> `roladdr`

`RRI`. `roladdr` ==> `src1`

`AAI`. `dest, destaddr, base, bound` の実効アドレス計算 ==> `aoladdr`

`EX`. 命令固有の動作
.op.(`src1, aoladdr`)

`CHK` では

.op.(`src1, aoladdr`) ==> `xreg`

`BFINS`, `BFINSU`, `BFCMP`, `BFCMPU` では、

.op.(`src1, aoladdr, width, src`)

`BFEXT`, `BFEXTU` では、

.op.(`src1, aoladdr, width`) ==> `dest`

`BFINS`, `BFEXT` の動作例は図402のとおり。

【0513】2オペランド `read~read` (RR)

該当命令:

364

`se` の実効アドレスを `A01`, `offset` を `R01` として扱う。`base` と `offset` の合成を行ない、操作対象となるビットを実際にアクセスするのは、`EX` の中で行なわれる。固定長ビットフィールド操作命令でも、ビット操作命令と同様に、`base` の実効アドレスを `A01`, `offset` を `R01` として扱う。`offset` と `base` 以外のオペランド、即ち `width`, `src`, `dest` については、`EX` の中でのみアクセスし、第 `n-read` オペランド (`R0n`)、第 `m-write` オペランド (`W0m`) としては扱わない。`CHK` の場合は、ビットパターンとの関係から、`index` を `R01`, `bound` を `A01` として扱い、`bound` の内容である (上限値: 下限値) の読みだしと `xreg` への書き込みは `EX` で行なうことにする。以上のような形で各命令の動作とモデルとの対応をつけておけば、命令ビットパターンやインプリメント方法とも矛盾しない。`AA1` では、`@SP+`, `@-SP` を指定することはできない。`src`, `offset`, `index` = `@SP+` の場合、`dest`, `destaddr`, `base`, `bound` の実効アドレス計算で参照される `SP` 値としては、インクリメント後の値が使用されることになる。`CHK` 命令の場合、これはアセンブラ表記での順序と逆になっているので、注意が必要である。また、固定長ビットフィールド命令で `offset` = `@SP+` を指定し、`src`, `width` などで同じ `SP` を指定した場合の `SP` 値としては、やはりインクリメント後の値が使用されることになる。アセンブラ表記では、`BFINS`, `BFCMP` の場合に `offset` よりも `src` の方が先に書かれるようになっているが、実際には `offset` の実効アドレス計算で更新された `SP` が `src` に反映されるので、注意が必要である。

【0512】

50

365
 CMP src1, src2
 CMPU src1, src2
 INDEX indexsize, subscript, xreg
 ACB step, xreg, limit, newpc
 SCB step, xreg, limit, newpc

CMP, CMPUでは、src1をR01に、src2をR02に対応させる。INDEXでは、indexsizeをR01に、subscriptをR02に対応させる。xregはEXで扱う。ACB, SCBではstepをR01に、limitをR02に対応させる。*

RA1. src1, indexsize, step のアドレス計算==> roladdr
 RB1. @roladdr==> src11
 RA2. src2, subscript, limitの実効アドレス計算==> ro2addr
 RB2. @ro2addr==> src21
 EX 命令固有の動作

.op.(src11, src21)

INDEX では

.op.(src11, src21, xreg) ==> xreg

ACB, SCBでは

.op.(src11, src21, xreg, newpc) ==> xreg

CMPの動作例は図403のとおり。INDEXの動作例は図404のとおり。ただし、INDEX命令については、インクリメントの制約によって、subscript=@SP+, xreg=SPの場合に必ずしもモデル通りの動作ができない場合がある。詳しくは2オペランドread~rmw(RM)の項を参照。ACB, SCBの動作例は図405のとおり。

【0515】2オペランドread~rmw(RM)

該当命令：

ADD src, dest
 ADDU src, dest
 ADDX src, dest

366
 *xreg, newpcはEXで扱う。したがって、src1, indexsize, step=@SP+の場合、src2, subscript, limitの実効アドレス計算で参照されるSPとしては、インクリメント後の値が使用される。また、INDEX, ACB, SCB命令でindexsize, step=@SP+またはsubscript, limit=@SP+の場合、xregとして参照されるSPとしては、やはりインクリメント後の値が使用される。

10 【0514】

SUB src, dest
 SUBU src, dest
 SUBX src, dest
 MUL src, dest
 MULU src, dest
 DIV src, dest
 DIVU src, dest
 REM src, dest
 REMU src, dest
 AND src, dest
 OR src, dest
 XOR src, dest
 SHA count, dest
 SHL count, dest
 ROT count, dest
 ADDDX src, dest
 SUBDX src, dest
 BSCH data, offset
 MULX src, dest, tmp
 DIVX src, dest, tmp

src, count, dataをR01に、dest, offsetをM01に対応させる。src=@SP+

50 の場合、destの実効アドレス計算で参照されるSP

367

としては、インクリメント後の値が使用される。MA1では、@SP+, @-SPのモードを指定することはできない。MULX, DIVXの場合、srcがR01、destがM01に対応し、tmpはEXの中で処理されるものとする。したがって、src=@SP+の場合に、destの実効アドレス計算で参照されるSP値、*

RA1. src, count, dataの実効アドレス計算==> roladdr
 RRI. @rlladdr==> src1
 MA1. dest, offsetの実効アドレス計算==> moladdr
 MR1. @moladdr==> dest1
 EX. 命令固有の動作

.op.(dest1, src1) ==> dest2

MULXでは

.op.(dest1, src1) ==> dest2, tmp

DIVXでは

.op.(dest1, src1, tmp)==>dest2, tmp

MW1. dest2 ==> @moladdr

ADDの動作例は図406のとおり。このうち、一つのハーフワードの中で、レジスタ指定RgR, RgMと汎用アドレッシング指定EaR, EaW, ShR, ShWの両方を含む命令では、EaR, EaW, ShR, ShWで@SP+, @-SPのモードを指定し、RgR, RgMでSPを指定した場合に、EaR, EaW, ShR, ShWによるSP値の更新がRgR, RgMとしてのSPの読みだし値に影響するため、パイプライン実装が難しいという意見がある。具体的に問題となるのは、以下のような命令である。

ADD:L @SP+, SP
 SUB:L @SP+, SP
 CMP:L @SP+, SP
 INDEX *. @SP+, SP
 (MOV:Lは含まない)

そこで、これらの命令については、動作がインプリメント依存になるものとする。つまり、ADD:L @SP+, SP実行後のSP値は、インプリメントによって不定の値をとるということにする。EITの検出も難しいので、EITとはしない。これらの命令は、短縮形が一般形と同じ動作をするという原則には違反することになる。

【0517】2オペランドaddress~address (AA)

368

*およびtmpで参照されるSP値としては、インクリメント後の値が使用される。また、tmpとdestで同じレジスタを指定した場合には、tmpの値が消え、最終的にdestの値が残ることになる。

【0516】

該当命令：

QINS entry, queue

entryをA01に、queueをA02に対応させる。キューリンクの実際の操作はEXのステップで行なわれる。

AA1. entryの実効アドレス計算==> aoladdr

AA2. queueの実効アドレス計算==> ao2addr

MA1~MR1. なし

EX. 命令固有の動作

.op.(aoladdr, ao2addr)

【0518】2オペランドaddress~read (AR)

該当命令：

CSI comp, update, dest

CSI命令では、ビットパターンとの関係から、destをA01に、updateをR01に対応させるcompのアクセスと比較、交換の実際の操作は、EXで行なわれる。CSI命令のオペランドの処理は、アセンブラ表記とは異なり、updateの実効アドレス計算、destの実効アドレス計算、comp値の参照といった順序で行なわれる。destでは@SP+, @-SPが使用できないが、updateでは@SP+が使用できるので、compでSPを参照した場合には注意が必要である。

369

370

RA1. updateの実効アドレス計算==> roladdr
 RR1. @roladdr==> update1
 AA1. destの実効アドレス計算==> aoladdr
 EX. 命令固有の動作
 .op.(aoladdr, update1.comp) ==> comp

CSIの動作例は図407のとおり。

【0519】1オペランドread~reglist
 (RL)

該当命令:

ENTER local, reglist
 EXITD reglist, adjsp

ENTER, EXITDでは、local, adjsp
 をR01に対応させる。reglistの参照とスタック
 フレームの操作は、EXのステップに含める。したがって*

RA1. local, adjspの実効アドレス計算==> roladdr
 (実際にはRnと#imm_dataのモードしか使用できないので、こ
 のステップでは何もアクセスしない。)

RR1. @roladdr==> src1

EX. reglist参照とスタックフレーム操作

ENTERでは

.op.(src1, R0~R13, FP, SP, reglist) ==> FP, SP, stack

EXITDでは

.op.(src1, FP, SP, stack, reglist) ==> R0~R13, FP, SP, PC

【0520】1オペランドaddress~reglist
 (AL)★

該当命令:

LDM src, reglist
 STM reglist, dest

LDM, STMでは、src, destをA01に対応
 させる。reglistの参照と実際のレジスタ転送
 は、EXのステップに含める。したがって、src, d
 estの実効アドレス計算の際に参照されるSP, F
 P, R0~R13は、レジスタ転送を始めてからの値で
 はなく、すべて命令実行前の値を使用することになる。
 LDM reglist, @SP+とSTM regl 40
 ist, @-SPについては、SPが複数回更新される
 ため、一般命令での@SP+, @-SPとは異なった扱
 いをする必要がある。すなわち、モデルの「実効アドレ
 ス計算」のステップで@SP+, @-SPによるSP値
 の更新を扱うのではなく、EXのステップでSP値の更
 新を扱うようにする。このため、SPをM01に対応さ
 せる。

[@SP+, @-SP以外のモードを使用した場合]

*つて、local, adjspの実効アドレス計算の際
 に参照されるSP, FP, R0~R13は、スタックフ
 レーム操作を始めてからの値ではなく、すべて命令実行
 10 前の値を使用することになる。EXITDでは、アセン
 ブラ表記でのオペランド順と逆の順番で実効アドレスが
 評価されるので、注意が必要である。(ただし、命令再
 実行の関係で、local, adjspはレジスタ直接
 Rnとイミディエート#imm_dataのモードしか
 使用できない。)

AA1. src, destの実効アドレス計算==> aoladdr
 EX. reglist参照とレジスタ転送

30

LDMでは

@(aoladdr ~...) ==> REG(reglist)

STMでは

REG(reglist) ==> @(aoladdr ~...)

【0521】[@SP+, @-SPのモードを使用した
 場合]

MR1. SP==> tmpaddr

EX. reglist参照とレジスタ転送

LDMでは

@(tmpaddr ~...) ==> REG(reglist)

tmpaddr + 転送レジスタ数 ==> tmpaddr

STMでは

tmpaddr - 転送レジスタ数 ==> tmpaddr

REG(reglist) ==> @(tmpaddr ~...)

[実際のインプリメントでは、レジスタ転送の順序は自
 由である。これと等価な動作をすればよい。]

MW1. tmpaddr ==> SP

50 LDM @SP+, reglist, の場合、regl

371

ist中にSPが指定されていても、最後のMW1のステップでtmpaddrがSPにoverwriteされるため、結果的にメモリからロードされたSP値が消えてしまうことになる。LDM, STMの動作例は図408のとおり。

【0522】付録13. キャッシュやTLBの整合性確保について

キャッシュやTLBの整合性の確保については、それぞれ関連する命令のところで説明を行なっているが、整理すると以下ようになる。

〔ATを変更した場合の整合性〕

・TLB, 論理キャッシュ(データキャッシュ)の整合性

—LDC, LDCTX, EIT, REITによりPSWの中のATが変更された場合、TLB, 論理データキャッシュの整合性が保証される。(LSIDがない場合はページされる。LSIDがある場合、AT=00の物理空間に対して特別なLSIDを与えると考えれば、必ずしもページする必要はない。)

・命令パイプライン、命令キャッシュの整合性

—命令コード整合性の状態は変化しない。ATを変更しても、命令コード整合性が保証されるわけではない。

【0523】〔UATB, SATBの操作をした場合の整合性〕

・TLB, 論理キャッシュ(データキャッシュ)の整合性

—LDC, LDCTXによるUATB, SATBの操作では、TLB, 論理データキャッシュの整合性が保証される。(LSIDがない場合、一般にはページされる)

・命令パイプライン、命令キャッシュの整合性

—LDC, LDCTXによりUATB, SATBを操作しても、命令コード整合性の状態は変化しない。インプリメントによっては、命令キャッシュのページによって命令コード整合性が良くなる場合もあるが、それが保証されているわけではない。例えば、論理空間A, 論理空間Bの命令コードをそれぞれ書き換えた後、論理空間AでPIB命令を実行すると、論理空間Aでの命令コード整合性は保証される。この後LDCまたはLDCTXでUATBを操作し、論理空間Bに切り換えたとしても、論理空間Bでの命令コード整合性は保証されず、それを保証するためには論理空間Bでもう一度PIB命令を実行する必要がある。ただし、論理空間BでPIB命令を実行してもしなくても、再度UATBを操作して論理空間Aに戻ってきた場合には、命令コードの整合性が保証されている。実際には、論理命令キャッシュのページにより、UATB操作の後には自動的に命令コード整合性が保証されるかもしれないが、プログラミング上はこの機能を当てにしていけない。将来LSIDを導入してページを避けることを考えると、一般には、UATBを操作しても命令コード整合性は変わらないと考えなければ

372

ならない。

【0524】〔ATEの操作をした場合の整合性〕

・TLB, 論理キャッシュ(データキャッシュ)の整合性

—LDATEによるATEの操作では、TLB, 論理データキャッシュの整合性が保証される。(影響のある部分がページされる)

—LDATEを用いず、一般のメモリアクセス命令でATEに使用しているメモリ領域を書き換えた場合には、TLB, 論理データキャッシュの整合性は保証されない。

・命令パイプライン、命令キャッシュの整合性

—ATEを更新した場合、そのATEによりアドレス変換される領域の「命令コードの整合性」は失われる。つまり、その領域のメモリ内容をプログラムとして実行しても、動作は保証されない。これは、LDATE命令を用いるかどうかには関係しない。命令コードの整合性を回復する必要があるれば、別にPIB命令を実行する。

【0525】〔メモリの操作をした場合の整合性〕

・論理キャッシュ(データキャッシュ)の整合性

—論理アドレスによってメモリをアクセスする場合には、論理データキャッシュの整合性が保証される。(キャッシュの制御機構による)

—LDP命令を使って、物理アドレスによりメモリアクセスする場合には、論理データキャッシュの整合性は保証されない。

・命令パイプライン、命令キャッシュの整合性

—メモリ内容を変更した場合、その領域の「命令コードの整合性」は失われる。これは、論理アドレスによるアクセスか物理アドレスによるアクセスかということには関係しない。内容を変更したメモリをプログラムとして実行するには、PIB命令を実行し、命令コードの整合性を回復する必要がある。

【0526】

【発明の効果】以上のようにこの発明によれば、2つのオペランド間の加算もしくは減算の算術演算を2の補数表現による符号付き二進数として行う命令を指定する第1の命令コードと、2つのオペランドの間の前記算術演算と同一の演算を絶対値表現された符号なし二進数として行う命令を指定する第2の命令コードとが用意された命令セットにより表現されるプログラムを解釈して、これら命令コードの各々の指定する算術演算を実行し、各演算によりデスティネーションオペランドに格納された演算結果におけるオーバーフローの発生を示す情報、もしくは各演算の演算結果の正負を示す情報をフラグで表現したので、加算もしくは減算の算術演算を符号付きで扱いたい符号なしで扱いたいかは、プログラムにおいて第1の命令コードを用いるか第2の命令コードを用いるかを選択するだけでよく、さらにそれぞれの演算結果のオーバーフローもしくは正負の状態を調べてその後

373

の処理内容を決定する際にはフラッグを参照するだけでよく、プログラム作成上の負担が少なくなるという効果がある。

【0527】また、2つのオペランドの間で2の補数表現による符号付き二進数としてその大小を比較する比較演算を指定する第1の命令コードと、2つのオペランドの間で絶対値表現された符号なし二進数としてその大小を比較する比較演算を指定する第2の命令コードとが用意された命令セットにより表現されるプログラムを解釈し、これら命令コードの各々の指定する比較演算を実行し、それぞれ実行された比較演算によるそれぞれの数学的大小関係の情報をフラッグで表現したので、比較演算を符号付きで扱いたい符号なしで扱いたいかは、プログラムにおいて第1の命令コードを用いるか第2の命令コードを用いるかを選択するだけでよく、さらにそれぞれの数学的大小関係を判断してその後の処理を行う場合でもフラッグを参照するだけでよく、プログラムの作成上の負担が少なくなるという効果がある。

【図面の簡単な説明】

【図1】本発明装置のレジスタセット説明図である。

【図2】本発明装置のレジスタセット説明図である。

【図3】本発明装置のビットについてのデータタイプ説明図である。

【図4】本発明装置のビットフィールドについてのデータタイプ説明図である。

【図5】本発明装置の符号なしビットフィールドについてのデータタイプ説明図である。

【図6】本発明装置の整数についてのデータタイプ説明図である。

【図7】本発明装置の整数についてのデータタイプ説明図である。

【図8】本発明装置の10進数についてのデータタイプ説明図である。

【図9】本発明装置の10進数についてのデータタイプ説明図である。

【図10】本発明装置のストリングについてのデータタイプ説明図である。

【図11】本発明装置のストリングについてのデータタイプ説明図である。

【図12】第8図は、本発明装置のキューについてのデータタイプ説明図である。

【図13】本発明装置の命令フォーマットの記述例を示す説明図である。

【図14】そのビットパターン図である。

【図15】本発明装置の命令フォーマット図である。

【図16】本発明装置の命令フォーマット図である。

【図17】本発明装置の命令フォーマット図である。

【図18】本発明装置の命令フォーマット図である。

【図19】本発明装置の命令フォーマット図である。

【図20】本発明装置の命令フォーマット図である。

374

【図21】本発明装置の命令フォーマット図である。

【図22】本発明装置の命令フォーマット図である。

【図23】本発明装置の命令フォーマット図である。

【図24】本発明装置の命令フォーマット図である。

【図25】本発明装置の命令フォーマット図である。

【図26】本発明装置のアドレッシングモードのフォーマット図である。

【図27】本発明装置のアドレッシングモードのフォーマット図である。

10 【図28】本発明装置のアドレッシングモードのフォーマット図である。

【図29】本発明装置のアドレッシングモードのフォーマット図である。

【図30】
・本発明装置のアドレッシングモードのフォーマット図である。

【図31】本発明装置のアドレッシングモードのフォーマット図である。

【図32】本発明装置のアドレッシングモードのフォーマット図である。

20 【図33】本発明装置のアドレッシングモードのフォーマット図である。

【図34】本発明装置のアドレッシングモードのフォーマット図である。

【図35】本発明装置のアドレッシングモードのフォーマット図である。

【図36】本発明装置のアドレッシングモードのフォーマット図である。

【図37】本発明装置のアドレッシングモードのフォーマット図である。

30 【図38】本発明装置のローカル変数配置例の説明図である。

【図39】本発明装置のアドレッシングモードのフォーマット図である。

【図40】本発明装置のアドレッシングモードのフォーマット図である。

【図41】本発明装置のアドレッシングモードのフォーマット図である。

【図42】本発明装置のアドレッシングモードのフォーマット図である。

40 【図43】命令MOVでの注意事項説明図である。

【図44】PSWのフォーマット図である。

【図45】PSSのフォーマット図である。

【図46】PSHのフォーマット図である。

【図47】命令セットの記述例を示すフォーマット図である。

【図48】命令MOVのフォーマット図及びそのフラグ変化の説明図である。

【図49】命令MOVUのフォーマット図である。

【図50】そのフラグ変化の説明図である。

50 【図51】命令PUSHのフォーマット図である。

375

【図52】そのフラッグ変化の説明図である。
 【図53】命令POPのフォーマット図である。
 【図54】そのフラッグ変化の説明図である。
 【図55】命令LDMのフォーマット図である。
 【図56】そのフラッグ変化の説明図である。
 【図57】ビットマップ指定の説明図である。
 【図58】命令STMのフォーマット図である。
 【図59】そのフラッグ変化の説明図である。
 【図60】ビットマップ指定の説明図である。
 【図61】ビットマップ指定の説明図である。
 【図62】命令MOVAのフォーマット図である。
 【図63】そのフラッグ変化の説明図である。
 【図64】命令PUSHAのフォーマット図である。
 【図65】そのフラッグ変化の説明図である。
 【図66】命令CMPのフォーマット図である。
 【図67】そのフラッグ変化の説明図である。
 【図68】命令CMPUのフォーマット図である。
 【図69】そのフラッグ変化の説明図である。
 【図70】命令CHKのフォーマット図である。
 【図71】そのフラッグ変化の説明図である。
 【図72】命令CHKのオペレーションの説明図である。
 【図73】命令ADDのフォーマット図である。
 【図74】そのフラッグ変化の説明図である。
 【図75】命令ADDUのフォーマット図である。
 【図76】そのフラッグ変化の説明図である。
 【図77】命令ADDXのフォーマット図である。
 【図78】そのフラッグ変化の説明図である。
 【図79】命令SUBのフォーマット図である。
 【図80】そのフラッグ変化の説明図である。
 【図81】命令SUBUのフォーマット図である。
 【図82】そのフラッグ変化の説明図である。
 【図83】命令SUBXのフォーマット図である。
 【図84】そのフラッグ変化の説明図である。
 【図85】命令MULのフォーマット図である。
 【図86】そのフラッグ変化の説明図である。
 【図87】命令MULUのフォーマット図である。
 【図88】そのフラッグ変化の説明図である。
 【図89】命令MULXのフォーマット図である。
 【図90】そのフラッグ変化の説明図である。
 【図91】命令DIVのフォーマット図である。
 【図92】そのフラッグ変化の説明図である。
 【図93】命令DIVUのフォーマット図である。
 【図94】そのフラッグ変化の説明図である。
 【図95】命令DIVXのフォーマット図である。
 【図96】そのフラッグ変化の説明図である。
 【図97】命令REMのフォーマット図である。
 【図98】そのフラッグ変化の説明図である。
 【図99】命令REMUのフォーマット図である。
 【図100】そのフラッグ変化の説明図である。

376

【図101】命令NEGのフォーマット図である。
 【図102】そのフラッグ変化の説明図である。
 【図103】命令INDEXのフォーマット図である。
 【図104】そのフラッグ変化の説明図である。
 【図105】命令ANDのフォーマット図である。
 【図106】そのフラッグ変化の説明図である。
 【図107】命令ORのフォーマット図である。
 【図108】そのフラッグ変化の説明図である。
 【図109】命令XORのフォーマット図である。
 10 【図110】そのフラッグ変化の説明図である。
 【図111】命令NOTのフォーマット図である。
 【図112】そのフラッグ変化の説明図である。
 【図113】命令SHAのフォーマット図である。
 【図114】そのフラッグ変化の説明図である。
 【図115】左シフトの説明図である。
 【図116】右シフトの説明図である。
 【図117】命令SHLのフォーマット図である。
 【図118】そのフラッグ変化の説明図である。
 【図119】左シフトの説明図である。
 20 【図120】右シフトの説明図である。
 【図121】命令ROTのフォーマット図である。
 【図122】そのフラッグ変化の説明図である。
 【図123】左回転の説明図である。
 【図124】右回転の説明図である。
 【図125】命令SHXLのフォーマット図である。
 【図126】そのフラッグ変化の説明図である。
 【図127】命令XHXLのフォーマット図である。
 【図128】命令XHXRのフォーマット図である。
 【図129】そのフラッグ変化の説明図である。
 30 【図130】命令SHXRのフォーマット図である。
 【図131】命令RVBYのフォーマット図である。
 【図132】そのフラッグ変化の説明図である。
 【図133】命令RVBIのフォーマット図である。
 【図134】そのフラッグ変化の説明図である。
 【図135】ビット操作命令の説明図である。
 【図136】ビット操作命令の説明図である。
 【図137】命令BTSTのフォーマット図である。
 【図138】そのフラッグ変化の説明図である。
 【図139】命令BSETのフォーマット図である。
 40 【図140】そのフラッグ変化の説明図である。
 【図141】命令BCLRのフォーマット図である。
 【図142】そのフラッグ変化の説明図である。
 【図143】命令BNOTのフォーマット図である。
 【図144】そのフラッグ変化の説明図である。
 【図145】命令BSCHのフォーマット図である。
 【図146】そのフラッグ変化の説明図である。
 【図147】固定長ビットフィールド操作命令の説明図である。
 50 【図148】ビットフィールド命令のフォーマット図である。

377

【図149】ビットフィールド命令のフォーマット図である。

【図150】命令BFEXTのフォーマット図である。

【図151】そのフラッグ変化の説明図である。

【図152】命令BFEXTUのフォーマット図である。

【図153】そのフラッグ変化の説明図である。

【図154】命令BFINSのフォーマット図である。

【図155】そのフラッグ変化の説明図である。

【図156】命令BFINSUのフォーマット図である。

【図157】そのフラッグ変化の説明図である。

【図158】命令BF CMPのフォーマット図である。

【図159】そのフラッグ変化の説明図である。

【図160】命令BF CMPUのフォーマット図である。

【図161】そのフラッグ変化の説明図である。

【図162】命令BVSCHのフォーマット図である。

【図163】そのフラッグ変化の説明図である。

【図164】命令BV MAPのフォーマット図である。

【図165】そのフラッグ変化の説明図である。

【図166】命令BV MAPのフォーマット図である。

【図167】命令BV MAPのフォーマット図である。

【図168】命令BV MAPのフォーマット図である。

【図169】命令BV CPYのフォーマット図である。

【図170】そのフラッグ変化の説明図である。

【図171】命令BV PATのフォーマット図である。

【図172】そのフラッグ変化の説明図である。

【図173】命令ADDDXのフォーマット図である。

【図174】そのフラッグ変化の説明図である。

【図175】命令SUBDXのフォーマット図である。

【図176】そのフラッグ変化の説明図である。

【図177】命令PACKssのフォーマット図である。

【図178】そのフラッグ変化の説明図である。

【図179】命令UNPKssのフォーマット図である。

【図180】そのフラッグ変化の説明図である。

【図181】命令UNPKssの説明図である。

【図182】命令SMOVのフォーマット図である。

【図183】そのフラッグ変化の説明図である。

【図184】命令SCMPの説明図である。

【図185】そのフラッグ変化の説明図である。

【図186】そのフラッグ変化の説明図である。

【図187】命令SSCHのフォーマット図である。

【図188】そのフラッグ変化の説明図である。

【図189】命令SSTRのフォーマット図である。

【図190】そのフラッグ変化の説明図である。

【図191】命令QINSのフォーマット図である。

【図192】そのフラッグ変化の説明図である。

378

【図193】命令QINSの説明図である。

【図194】命令QINSの説明図である。

【図195】命令QINSの説明図である。

【図196】命令QDELのフォーマット図である。

【図197】そのフラッグ変化の説明図である。

【図198】命令QDELの説明図である。

【図199】命令QDELの説明図である。

【図200】命令QDELの説明図である。

【図201】命令QSCHのフォーマット図である。

【図202】そのフラッグ変化の説明図である。

【図203】命令QSCHの説明図である。

【図204】命令QSCHの説明図である。

【図205】命令QSCHの説明図である。

【図206】命令BRAのフォーマット図である。

【図207】そのフラッグ変化の説明図である。

【図208】命令Bccのフォーマット図である。

【図209】そのフラッグ変化の説明図である。

【図210】命令BSRのフォーマット図である。

【図211】そのフラッグ変化の説明図である。

【図212】命令JMPのフォーマット図である。

【図213】そのフラッグ変化の説明図である。

【図214】命令JSRのフォーマット図である。

【図215】そのフラッグ変化の説明図である。

【図216】命令ACBのフォーマット図である。

【図217】そのフラッグ変化の説明図である。

【図218】命令ACBの説明図である。

【図219】命令SCBのフォーマット図である。

【図220】そのフラッグ変化の説明図である。

【図221】命令ENTERのフォーマット図である。

【図222】そのフラッグ変化の説明図である。

【図223】命令ENTERの説明図である。

【図224】命令EXITDの説明図である。

【図225】そのフラッグ変化の説明図である。

【図226】命令EXITDの説明図である。

【図227】命令RTSのフォーマット図である。

【図228】そのフラッグ変化の説明図である。

【図229】命令NOPのフォーマット図である。

【図230】そのフラッグ変化の説明図である。

【図231】命令PIBのフォーマット図である。

【図232】そのフラッグ変化の説明図である。

【図233】命令BSETIのフォーマット図である。

【図234】そのフラッグ変化の説明図である。

【図235】命令BCLR Iのフォーマット図である。

【図236】そのフラッグ変化の説明図である。

【図237】命令CSIのフォーマット図である。

【図238】そのフラッグ変化の説明図である。

【図239】命令LDCのフォーマット図である。

【図240】そのフラッグ変化の説明図である。

【図241】命令STCのフォーマット図である。

【図242】そのフラッグ変化の説明図である。

379

【図243】 命令LDPSBのフォーマット図である。
 【図244】 そのフラグ変化の説明図である。
 【図245】 命令LDPSMのフォーマット図である。
 【図246】 そのフラグ変化の説明図である。
 【図247】 命令STPSBのフォーマット図である。
 【図248】 そのフラグ変化の説明図である。
 【図249】 命令STPSMのフォーマット図である。
 【図250】 そのフラグ変化の説明図である。
 【図251】 命令LDPのフォーマット図である。
 【図252】 そのフラグ変化の説明図である。
 【図253】 命令STPのフォーマット図である。
 【図254】 そのフラグ変化の説明図である。
 【図255】 命令J RNGのフォーマット図である。
 【図256】 そのフラグ変化の説明図である。
 【図257】 命令J RNGのフォーマット図である。
 【図258】 命令J RNGのフォーマット図である。
 【図259】 命令J RNGのフォーマット図である。
 【図260】 命令J RNGのフォーマット図である。
 【図261】 命令J RNGのフォーマット図である。
 【図262】 命令J RNGのフォーマット図である。
 【図263】 命令RRNGのフォーマット図である。
 【図264】 そのフラグ変化の説明図である。
 【図265】 命令RRNGの説明図である。
 【図266】 命令RRNGの説明図である。
 【図267】 命令RRNGの説明図である。
 【図268】 命令TRAPAのフォーマット図である。
 【図269】 そのフラグ変化の説明図である。
 【図270】 命令TRAPのフォーマット図である。
 【図271】 そのフラグ変化の説明図である。
 【図272】 命令REITのフォーマット図である。
 【図273】 そのフラグ変化の説明図である。
 【図274】 命令REITのフォーマット図である。
 【図275】 命令WAITのフォーマット図である。
 【図276】 そのフラグ変化の説明図である。
 【図277】 命令LDCTXのフォーマット図である。
 【図278】 そのフラグ変化の説明図である。
 【図279】 命令STCTXのフォーマット図である。
 【図280】 そのフラグ変化の説明図である。
 【図281】 命令ACSのフォーマット図である。
 【図282】 そのフラグ変化の説明図である。
 【図283】 命令MOVPAのフォーマット図である。
 【図284】 そのフラグ変化の説明図である。
 【図285】 命令MOVPAのフォーマット図である。
 【図286】 命令MOVPAのフォーマット図である。
 【図287】 命令LDATEの説明図である。
 【図288】 そのフラグ変化の説明図である。
 【図289】 そのフラグ変化の説明図である。
 【図290】 命令STATEのフォーマット図である。
 【図291】 そのフラグ変化の説明図である。
 【図292】 そのフラグ変化の説明図である。

380

【図293】 命令PTLBのフォーマット図である。
 【図294】 そのフラグ変化の説明図である。
 【図295】 命令PSTLBのフォーマット図である。
 【図296】 そのフラグ変化の説明図である。
 【図297】 ATフィールドの説明図である。
 【図298】 論理アドレスの説明図である。
 【図299】 ページアドレスの説明図である。
 【図300】 多重論理空間の説明図である。
 【図301】 UATBのフォーマット図である。
 10 【図302】 SATBのフォーマット図である。
 【図303】 SXの制限を示す説明図である。
 【図304】 テーブル領域の説明図である。
 【図305】 テーブル領域の説明図である。
 【図306】 テーブル領域の説明図である。
 【図307】 STEのフォーマット図である。
 【図308】 PXの制限を示す説明図である。
 【図309】 STEの説明図である。
 【図310】 論理アドレスの説明図である。
 【図311】 PTEのフォーマット図である。
 20 【図312】 PTEの各値とアクセス可能リングとの関係を示す説明図である。
 【図313】 PTEの説明図である。
 【図314】 本発明装置の論理アドレス拡張に係るメモリマップの図である。
 【図315】 本発明装置の論理アドレス拡張に係るメモリマップの図である。
 【図316】 データ転送命令のフラグ変化の説明図である。
 【図317】 比較テスト命令のフラグ変化の説明図である。
 30 【図318】 算術演算命令のフラグ変化の説明図である。
 【図319】 論理演算命令のフラグ変化の説明図である。
 【図320】 シフト命令のフラグ変化の説明図である。
 【図321】 ビット操作命令のフラグ変化の説明図である。
 【図322】 固定長ビットフィールド命令のフラグ変化の説明図である。
 40 【図323】 固定長ビットフィールド命令のフラグ変化の説明図である。
 【図324】 任意長ビットフィールド命令のフラグ変化の説明図である。
 【図325】 10進演算命令のフラグ変化の説明図である。
 【図326】 スtring命令のフラグ変化の説明図である。
 【図327】 キュー操作命令のフラグ変化の説明図である。
 50

381

【図328】ジャンプ命令のフラッグ変化の説明図である。

【図329】マルチプロセッサ命令のフラッグ変化の説明図である。

【図330】制御空間、物理空間操作命令のフラッグ変化の説明図である。

【図331】OS関連命令のフラッグ変化の説明図である。

【図332】MMU関連命令のフラッグ変化の説明図である。

【図333】サブルーチンコールの説明図である。

【図334】スタックフレームの説明図である。

【図335】命令シーケンスの説明図である。

【図336】命令シーケンスの説明図である。

【図337】プログラム例を示す説明図である。

【図338】サブルーチンコールの説明図である。

【図339】制御空間の説明図である。

【図340】PSWのフォーマット図である。

【図341】IMASKのフォーマット図である。

【図342】SMRNGのフォーマット図である。

【図343】CTXBBのフォーマット図である。

【図344】DIのフォーマット図である。

【図345】CSWのフォーマット図である。

【図346】DCEのフォーマット図である。

【図347】CTXBFMのフォーマット図である。

【図348】EITVBのフォーマット図である。

【図349】JRNGVBのフォーマット図である。

【図350】SP0~SP3のフォーマット図である。

【図351】SPIのフォーマット図である。

【図352】IOADDR, IOMASKのフォーマット図である。

【図353】UATBのフォーマット図である。

【図354】SATBのフォーマット図である。

【図355】LSIDのフォーマット図である。

【図356】CTXBのフォーマット図である。

【図357】CTXBFMのフォーマット図である。

【図358】EITVTEのフォーマット図である。

【図359】スタックフレームの説明図である。

【図360】EITのスタックフォーマット図である。

【図361】EITのスタックフォーマット図である。

【図362】IOINFのフォーマット図である。

【図363】EITのベクトルテーブルの図である。

【図364】EITのベクトルテーブルの図である。

【図365】JRNGの説明図である。

【図366】EITの説明図である。

【図367】EITの説明図である。

【図368】IMASKの説明図である。

【図369】システムコールの説明図である。

【図370】システムコールの説明図である。

【図371】DCEの説明図である。

382

【図372】DCE, DI, EIの比較図である。

【図373】DCEの使用例の説明図である。

【図374】ビット割当図である。

【図375】ビット割当図である。

【図376】ビット割当図である。

【図377】ビット割当図である。

【図378】ビット割当図である。

【図379】ビット割当図である。

【図380】ビット割当図である。

【図381】ビット割当図である。

【図382】ビット割当図である。

【図383】ビット割当図である。

【図384】ビット割当図である。

【図385】ビット割当図である。

【図386】オペランドフィールド名索引図である。

【図387】オペランドフィールド名索引図である。

【図388】オペランドフィールド名索引図である。

【図389】ccccの割当て図である。

【図390】eeeeの割当て図である。

【図391】M__flagの説明図である。

【図392】BVMA P命令の演算コード図である。

【図393】アドレッシングモードの対応図である。

【図394】アドレッシングモードの対応図である。

【図395】アドレッシングモードの対応図である。

【図396】命令実行モデルのオペランドとの対応図である。

【図397】PUSHA @ SP等の説明図である。

【図398】POP命令の説明図である。

【図399】MOV命令の説明図である。

【図400】PACK命令等の説明図である。

【図401】STC命令等の説明図である。

【図402】BFINS命令等の説明図である。

【図403】CMP命令の説明図である。

【図404】INDEX命令の説明図である。

【図405】ACB命令等の説明図である。

【図406】ADD命令の説明図である。

【図407】CSI命令の説明図である。

【図408】LDM命令等の説明図である。

【図409】四則演算命令の動作説明図である。

【図410】四則演算命令の動作説明図である。

【図411】四則演算命令の動作説明図である。

【図412】四則演算命令の動作説明図である。

【図413】四則演算命令の動作説明図である。

【図414】四則演算命令の動作説明図である。

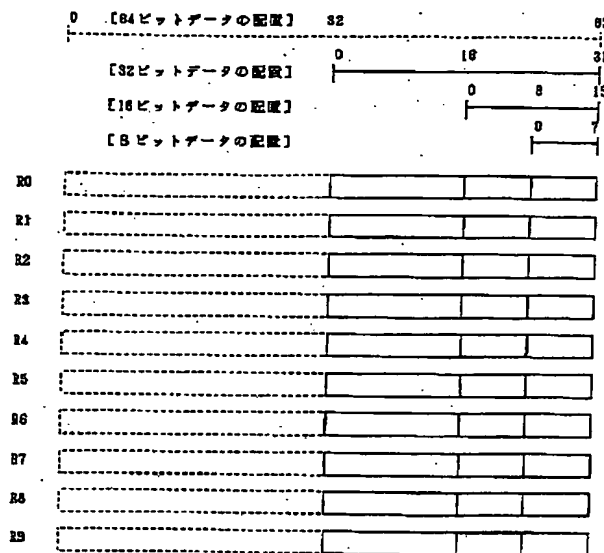
【図415】オーバーフロー時のフラッグ説明図である。

【図416】オーバーフロー時のフラッグ説明図である。

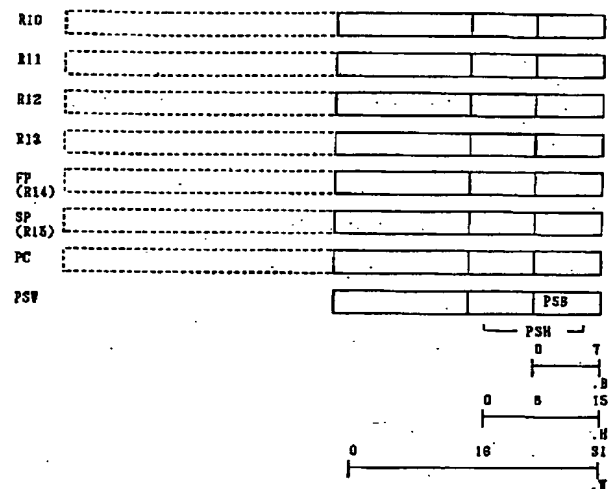
【図417】オーバーフロー時のフラッグ説明図である。

る。

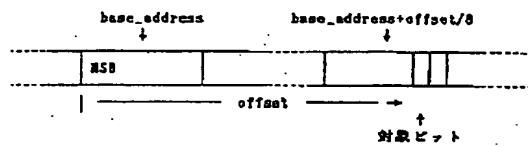
【図1】



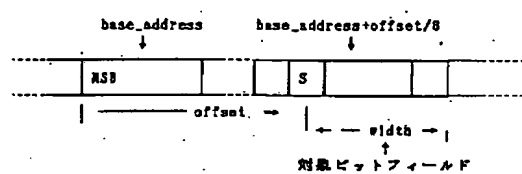
【図2】



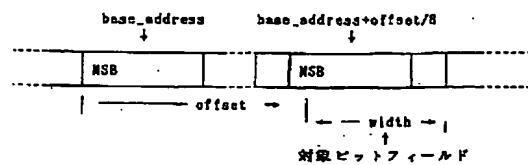
【図3】



【図4】



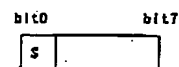
【図5】



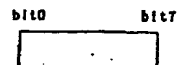
【図6】

・符号付き8ビット整数

S:符号ビット



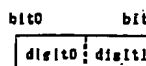
・符号なし8ビット整数



【図8】

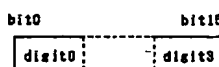
・1バイト(2桁)符号なしBCD数

digit0の方が上位桁となる



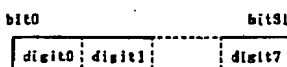
・2バイト(4桁)符号なしBCD数

digit0の方が上位桁となる



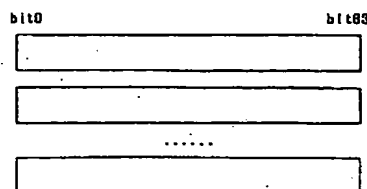
・4バイト(8桁)符号なしBCD数

digit0の方が上位桁となる



【図11】

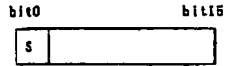
・64ビットデータのストリング <L>



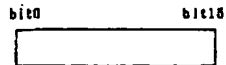
【図7】

- ・符号付き16ビット整数

S:符号ビット

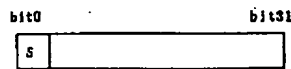


- ・符号なし16ビット整数

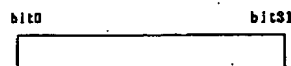


- ・符号付き32ビット整数

S:符号ビット



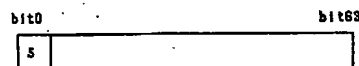
- ・符号なし32ビット整数



- ・符号付き64ビット整数

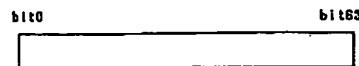
<<L1>>

S:符号ビット

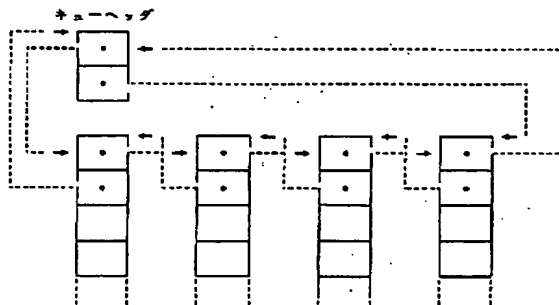


- ・符号なし64ビット整数

<<L1>>



【図12】



【図14】

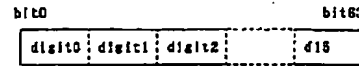
MOV:1 01001000 11 ShY ShY
 01001000 1110000 00000000 00010010
 <アドレス> +0 +1 +2 +3

【図9】

- ・8バイト(16桁)符号なしBCD数

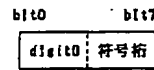
<<L1>>

digit0の方が上位桁となる



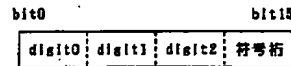
- ・1バイト(2桁)符号付きBCD数

<<L2>>



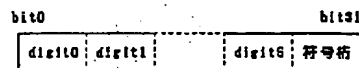
- ・2バイト(4桁)符号付きBCD数

<<L2>>



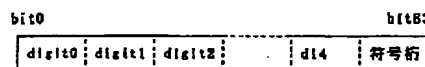
- ・4バイト(8桁)符号付きBCD数

<<L2>>



- ・8バイト(16桁)符号付きBCD数

<<L2>>

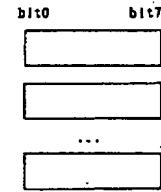


- ・多倍長BCD数

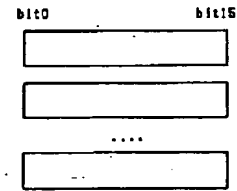
<<コプロセッサ>>

【図10】

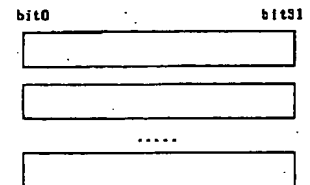
- ・8ビットデータのストリング



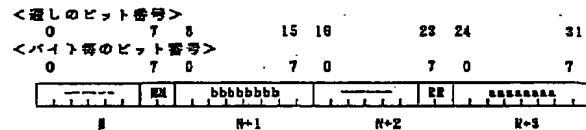
- ・16ビットデータのストリング



- ・32ビットデータのストリング



【図13】



<アドレス>

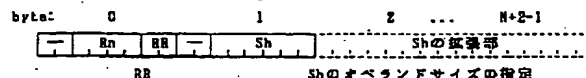
←低アドレス
←MSB側

高アドレス→
LSB側→

→命令を読む方向→

【図15】

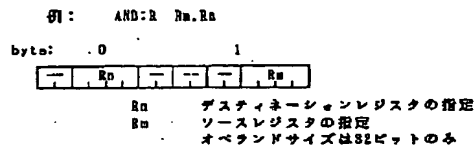
例: MOV:S Rn,Sh S-format
 MOV:L Sh,Rn L-format



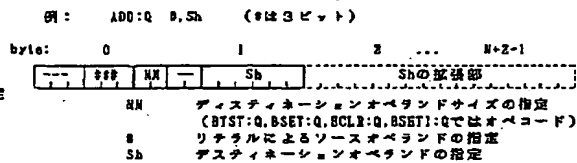
Shのオペランドサイズの指定
 レジスタ上に置かれたもう一方のオペ
 ランドのサイズは、32ビットに固定されて
 いる。

Sh
 Rn
 ソース(デスティネーション)オペラ
 ンドの指定
 デスティネーション(ソース)レジスタ
 の指定

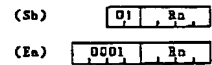
【図16】



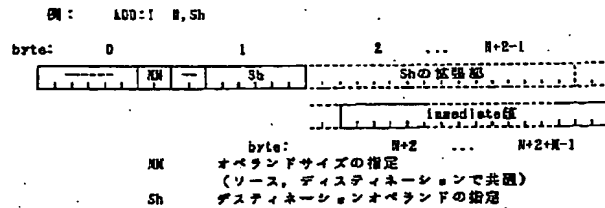
【図17】



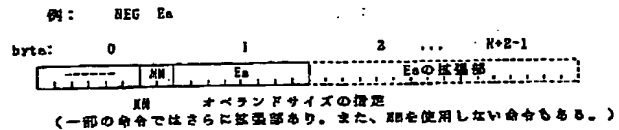
【図26】



【図18】

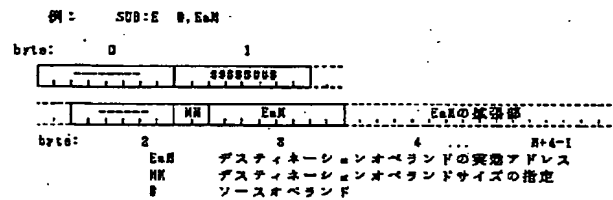
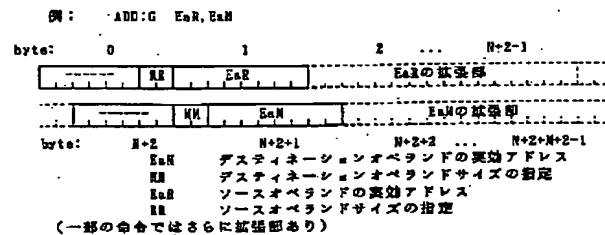


【図19】

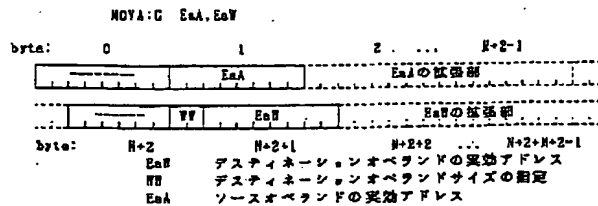


【図21】

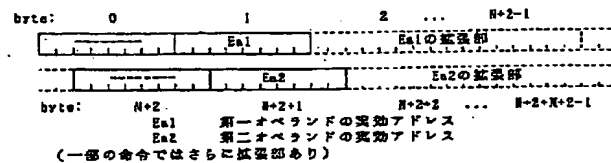
【図20】



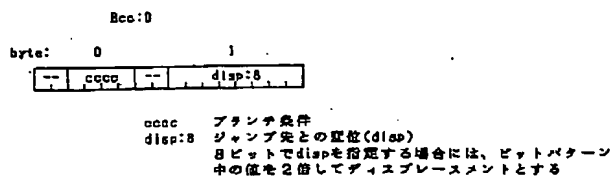
【図22】



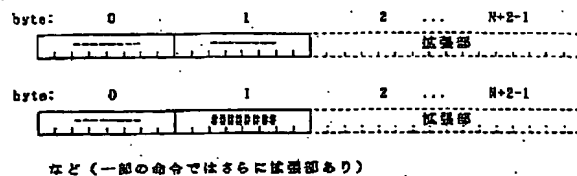
【図23】



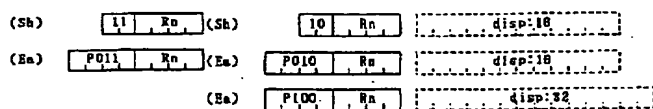
【図24】



【図25】

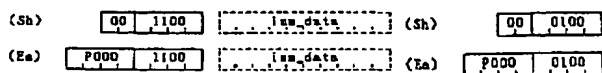


【図27】



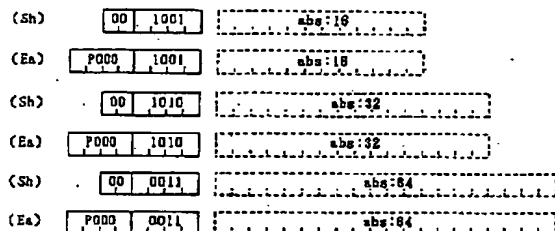
【図28】

【図29】

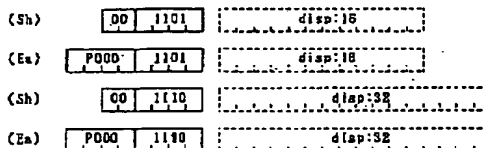


【図32】

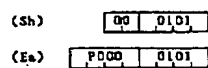
【図30】



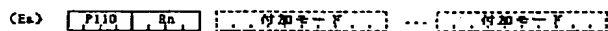
【図31】



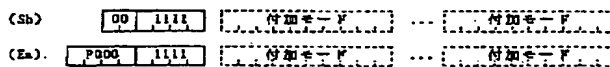
【図33】



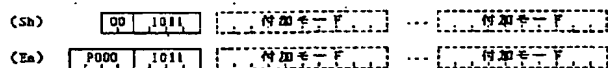
【図34】



【図35】

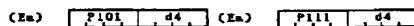


【図36】

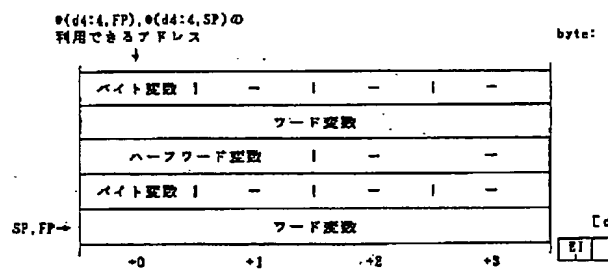


【図37】

【図39】



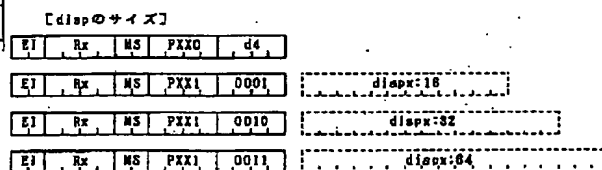
【図38】



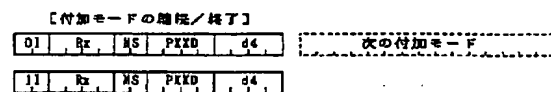
【図40】



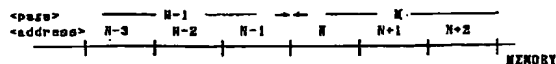
【図42】



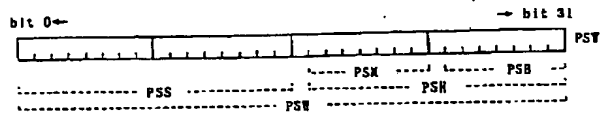
【図41】



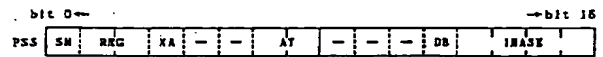
【図43】



【図44】



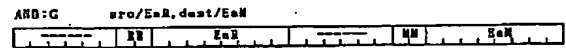
【図45】



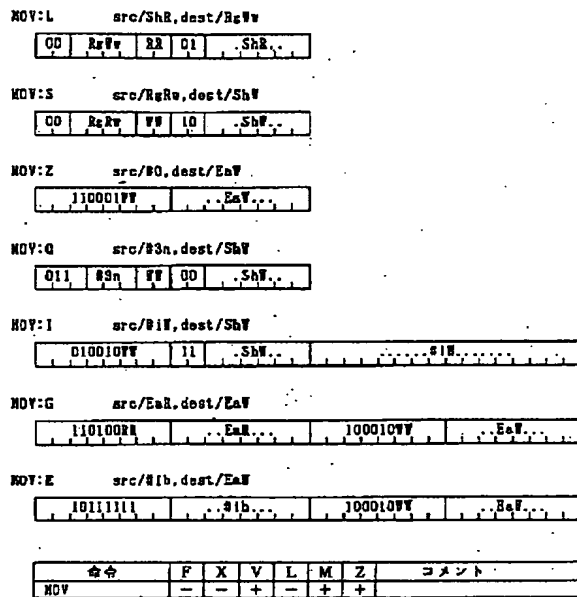
【図46】



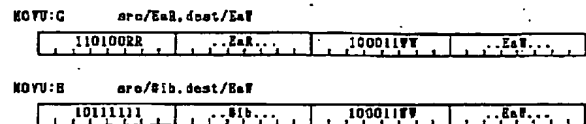
【図47】



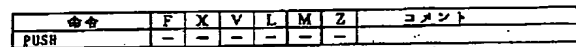
【図48】



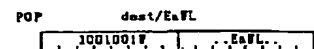
【図49】



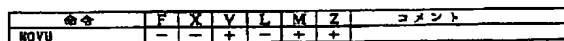
【図52】



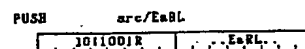
【図53】



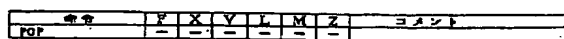
【図50】



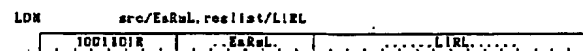
【図51】



【図54】



【図55】



【図56】

命令	F	X	V	L	M	Z	コメント
LDN	-	-	-	-	-	-	

【図57】

MSB←	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	→LSB
[bit位置] [19~29]	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	

【図58】

STN	reg1st/LsL,dest/EsVsl
1000101W	...EaVsl...LsL...

【図59】

命令	F	X	V	L	M	Z	コメント
STN	-	-	-	-	-	-	

【図60】

MSB←	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	→LSB
[bit位置] [19~29]	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31	

【図61】

MSB←	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	→LSB
[bit位置] [19~29]	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	

【図62】

NOVA:R	srcaddr/0(0d18, RsrP),dest/RsrP
00 RsrP 11 1100 RsrP	...0d18...
NOVA:G	srcaddr/EaA,dest/EaV
1101000	...EaA...101101W...EaV...

【図63】

命令	F	X	V	L	M	Z	コメント
NOVA	-	-	-	-	-	-	

【図64】

PUSHA	srcaddr/EaA
1010001S	...EaA...

【図65】

命令	F	X	V	L	M	Z	コメント
PUSHA	-	-	-	-	-	-	

【図66】

CMP:L	src1/ShR,src2/RsRw
00 RsRw RR 00	...ShR...
CMP:Z	src1/00,src2/EaR11
110000SS	...EaR11...
CMP:Q	src1/R3n,src2/ShR11
010 R3n RR 00	...ShR11...
CMP:I	src1/R1R,src2/ShR11
010000RR 11	...ShR11...R1R...
CMP:G	src1/EaR,src2/EaR11
110100RR	...EaR...100000SS...EaR11...
CMP:E	src1/R1b,src2/EaR11
10111111	...R1b...100000SS...EaR11...

【図67】

命令	F	X	V	L	M	Z	コメント
CMP	-	-	-	+	-	+	

【図68】

CMPU:G	src1/EaR,src2/EaR11
110100RR	...EaR...100001SS...EaR11...
CMPU:E	src1/R1b,src2/EaR11
10111111	...R1b...100001SS...EaR11...

【図69】

命令	F	X	V	L	M	Z	コメント
CMPU	-	-	-	+	-	+	

【図70】

命令	F	X	V	L	M	Z	コメント
CHK	-	+	+	+	+	+	L,2は下限値との比較

bound/EaRdR, index/EaR, xreg/RgVR
 110101RR ..EaR... 00 RgVR 1c ..EaRdR...
 c 下限値を引くかどうかの選択
 c=0 下限値を引かない (/N)
 c=1 下限値を引く (/S)
 RR 上限値、下限値、比較値のサイズ
 (上限値：下限値) の有効アドレス
 bound 比較値の実効アドレス
 index 比較値をロードするレジスタ
 xreg 比較値をロードするレジスタ

【図71】

命令	F	X	V	L	M	Z	コメント
CHK	-	+	+	+	+	+	L,2は下限値との比較

【図72】

	index < 下限値	下限値 ≤ index index < 上限値	上限値 ≤ index
L_flag	1	0	00
V_flag	1	0	1

【図73】

ADD:L	src/ShRv,dest/RgWv
10	RgWv 01 00 ..ShRv...
ADD:Q	src/#3n,dest/ShM
010	#3n NM 01 ..ShM...
ADD:I	src/#1M,dest/ShM
010001MM	11 ..ShM... ..#1M...
ADD:G	src/EaR,dest/EaM
110100RR	..EaR... 000000MM ..EaM...
ADD:E	src/#1b,dest/EaM
10111111	..#1b... 000000MM ..EaM...

【図74】

命令	F	X	V	L	M	Z	コメント
ADD	-	+	+	+	+	+	

【図75】

ADDG:C	src/EaR,dest/EaM
110100RR	..EaR... 000001MM ..EaM...
ADDU:E	src/#1b,dest/EaM
10111111	..#1b... 000001MM ..EaM...

【図76】

命令	F	X	V	L	M	Z	コメント
ADDU	-	+	+	0	+	+	

【図77】

ADDG:C	src/EaR,dest/EaM
110100RR	..EaR... 000100MM ..EaM...
ADDX:E	src/#1b,dest/EaM
10111111	..#1b... 000100MM ..EaM...

【図78】

命令	F	X	V	L	M	Z	コメント
ADDX	-	+	+	+	+	+	

【図80】

命令	F	X	V	L	M	Z	コメント
SUB	-	+	+	+	+	+	

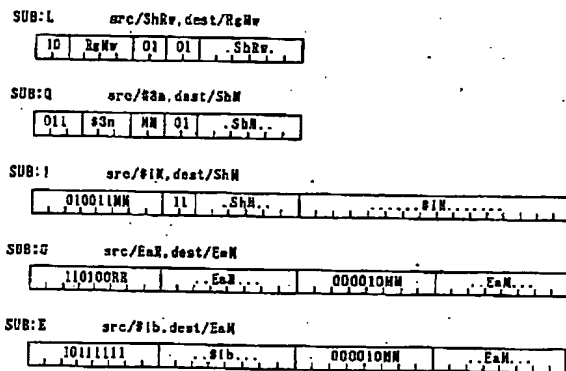
【図82】

命令	F	X	V	L	M	Z	コメント
SUBU	-	+	+	+	+	+	

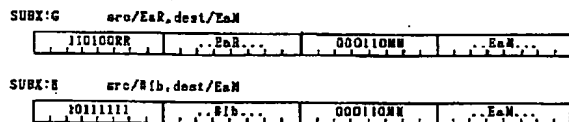
【図84】

命令	F	X	V	L	M	Z	コメント
SUBX	-	+	+	+	+	+	

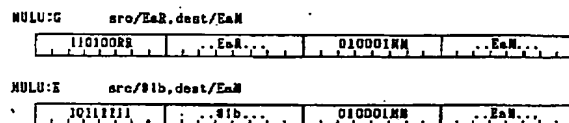
【図79】



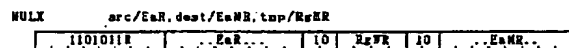
【図83】



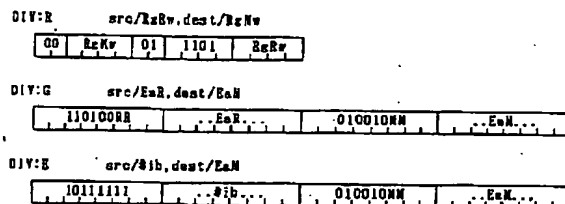
【図87】



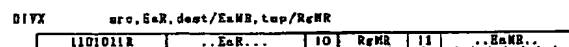
【図89】



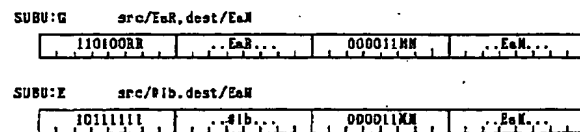
【図91】



【図95】



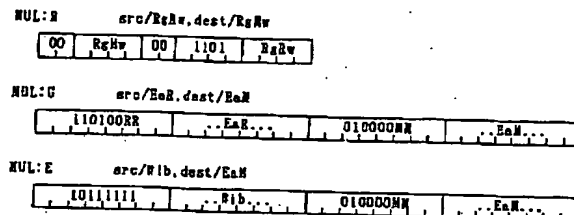
【図81】



【図86】

命令	F	X	V	L	M	Z	コメント
MUL	-	-	+	+	+	+	

【図85】



【図88】

命令	F	X	V	L	M	Z	コメント
MULU	-	-	+	0	+	+	

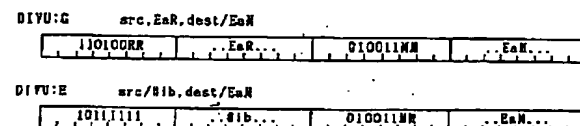
【図90】

命令	F	X	V	L	M	Z	コメント
MULX	*	-	0	0	+	+	R, Zはdestレジスタ, Fはtmp=0

【図92】

命令	F	X	V	L	M	Z	コメント
DIV	-	-	0	+	+	+	-(最小負数)÷(-1)の場合 -0除算, INTの場合
	-	-	1	0	1	0	
	-	-	1	-	-	-	

【図93】



【図94】

命令	F	X	V	L	M	Z	コメント
DIV	-	-	0	0	+	+	-0除算, EITの場合

【図96】

命令	F	X	V	L	M	Z	コメント
DIV	*	-	0	0	+	+	M,Zはdest基準, Fはtmp=0 -dest?h'-78-の場合 -0除算, EITの場合

【図97】

REM:G	src/EaR,dest/EaM		
	110100RR	..EaR..	01010MM ..EaM..

REM:E	src/#1b,dest/EaM		
	10111111	..#1b..	01010MM ..EaM..

【図98】

命令	F	X	V	L	M	Z	コメント
REM	-	-	0	0	+	+	-0除算, EITの場合

【図100】

【図99】

REM:G	src/EaR,dest/EaM		
110100RR	..EaR..	01011MM	..EaM..

REM:E	src/#1b,dest/EaM		
10111111	..#1b..	01011MM	..EaM..

命令	F	X	V	L	M	Z	コメント
REM	-	-	0	0	+	+	-0除算, EITの場合

【図111】

NOT	dest/EaM
110011MM	..EaM...

【図101】

NEG	dest/EaM
110010MM	..EaM...

【図102】

命令	F	X	V	L	M	Z	コメント
NEG	-	-	+	+	+	+	

【図103】

INDEX	indexsize/EaR,subscript/EaR,xreg/RgM				
1101011R	..EaR...	11	RgM	SS	..EaR2..

R xregとindexsizeのサイズ
 R=0 32ビット
 R=1 64ビット <<LX>>
 SS subscriptのサイズ
 xreg アドレス計算用レジスタ

【図104】

命令	F	X	V	L	M	Z	コメント
INDEX	-	-	+	+	+	+	M,Zはxreg基準

【図106】

命令	F	X	V	L	M	Z	コメント
AND	-	-	-	-	+	+	

【図105】

AND:R	src/RgR,dest/RgR				
	00	RgR	00	1100	RgR

AND:I	src/#1N,dest/ShM				
	01010MM	11	ShM		ShM

AND:G	src/EaR,dest/EaM				
	110100RR	EaR	00100MM	EaM	

AND:E	src/#1b,dest/EaM				
	10111111	#1b	00100MM	EaM	

【図107】

OR:R	src/RgR,dest/RgR				
	00	RgR	01	1100	RgR

OR:I	src/#1N,dest/ShM				
	010101MM	11	ShM		#1N

OR:G	src/EaR,dest/EaM				
	110100RR	EaR	001001MM	EaM	

OR:E	src/#1b,dest/EaM				
	10111111	#1b	001001MM	EaM	

【図108】

命令	F	X	V	L	M	Z	コメント
OR	-	-	-	-	+	+	

【図110】

命令	F	X	V	L	M	Z	コメント
XOR	-	-	-	-	+	+	

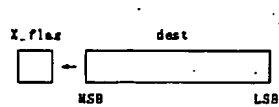
【図112】

命令	F	X	V	L	M	Z	コメント
NOT	-	-	-	-	+	+	

【図114】

命令	F	X	V	L	M	Z	コメント
SHA	-	+	+	+	+	+	

【図115】



【図118】

命令	F	X	V	L	M	Z	コメント
SHL	-	+	-	-	+	+	

【図126】

命令	F	X	V	L	M	Z	コメント
SHXL	-	+	-	-	+	+	

【図121】

命令	F	X	V	L	M	Z	コメント
ROT:G	-	+	-	-	+	+	
ROT:E	-	+	-	-	+	+	

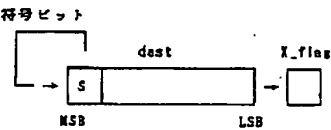
【図109】

XOR:R	src/RgRv,dest/RgRv
XOR:I	src/RiM,dest/ShM
XOR:G	src/EaR,dest/EaM
XOR:E	src/Rib,dest/EaM

【図113】

SHA:C	count/RSc,dest/ShM (右シフト、count<0)
SHA:G	count/EaR,dest/EaM
SHA:E	count/Rib,dest/EaM

【図116】

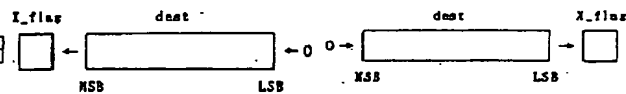


【図117】

SRL:Q	count/RSn,dest/ShM (左シフト、count>0)
SRL:C	count/RSc,dest/ShM (右シフト、count<0)
SRL:G	count/EaR,dest/EaM
SRL:E	count/Rib,dest/EaM

【図119】

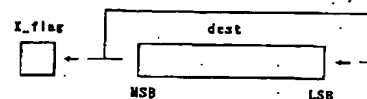
【図120】



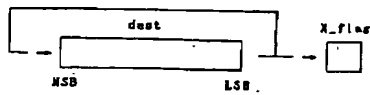
【図122】

命令	F	X	V	L	M	Z	コメント
ROT	-	+	-	-	+	+	

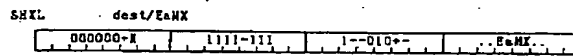
【図123】



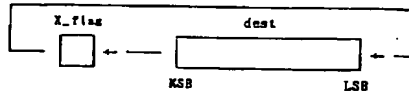
【図124】



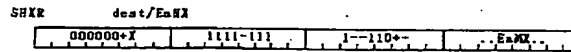
【図125】



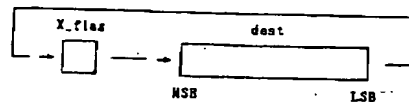
【図127】



【図128】



【図130】



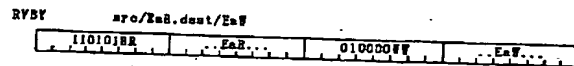
【図129】

命令	F	X	V	L	M	Z	コメント
SHL	-	+	-	-	+	+	

【図132】

命令	F	X	V	L	M	Z	コメント
RVBY	-	-	-	-	-	-	

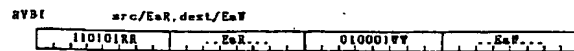
【図131】



【図134】

命令	F	X	V	L	M	Z	コメント
RVBI	-	-	-	-	-	-	

【図133】

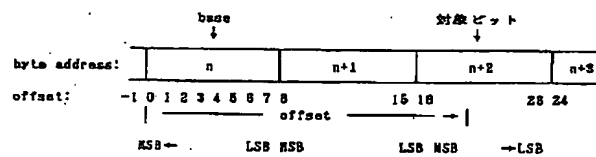


【図136】

	レジスタ対象	メモリ対象
アクセスサイズ.B	OK	OK
アクセスサイズ.H	OK	OK<<L2>>
アクセスサイズ.W	OK	OK<<L2>>
アクセスサイズ.L	<<L2>>	<<L2>>

アセンブラでのデフォルトはすべて.B

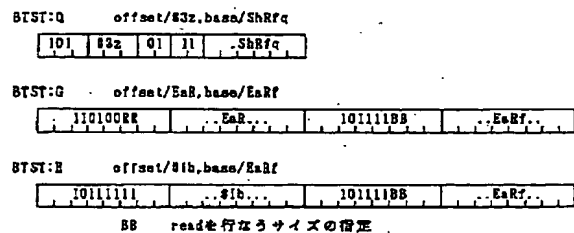
【図135】



【図138】

命令	F	X	V	L	M	Z	コメント
BTST	-	-	-	-	+	+	Zがテスト結果

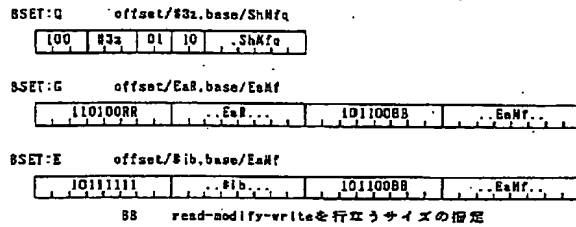
【図137】



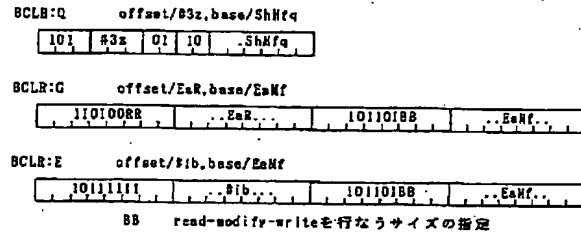
【図140】

命令	F	X	V	L	M	Z	コメント
BSET	-	-	-	-	-	+	Zがテスト結果

【図139】



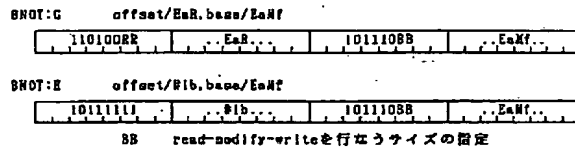
【図141】



【図142】

命令	F	X	V	L	M	Z	コメント
BCLR	-	-	-	-	-	+	Zがテスト結果

【図143】



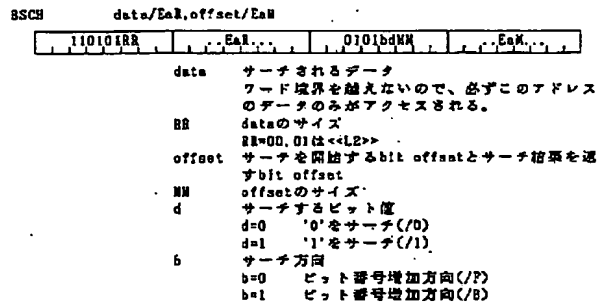
【図144】

命令	F	X	V	L	M	Z	コメント
BNOT	-	-	-	-	-	+	Zがテスト結果

【図146】

命令	F	X	V	L	M	Z	コメント
BSCN	-	-	+	-	-	-	Yはサーチ失敗

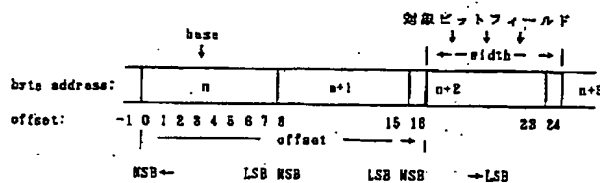
【図145】



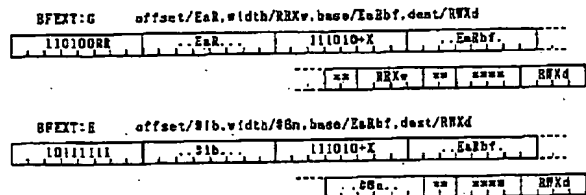
【図151】

命令	F	X	V	L	M	Z	コメント
BFEXT	-	-	+	-	+	+	

【図147】



【図150】



【図153】

命令	F	X	V	L	M	Z	コメント
BFEXTU	-	-	+	-	+	+	

【図155】

命令	F	X	V	L	M	Z	コメント
BFINS	-	-	+	-	+	+	

【図148】

BFINS:G:I.W	#src[.BHWL],	offset[.BHWL],	width[.W],	base[.W]
BFINS:G:R.W	Rs[.W],	offset[.BHWL],	width[.W],	base[.W]
BFINS:G:I.L	#src[.BHWL],	offset[.BHWL],	width[.L],	base[.L]
BFINS:G:R.L	Rs[.L],	offset[.BHWL],	width[.L],	base[.L]
BFINS:E:I.W	#src[.BHWL],	#offset,	#width,	base[.W]
BFINS:E:R.W	Rs[.W],	#offset,	#width,	base[.W]
BFINS:E:I.L	#src[.BHWL],	#offset,	#width,	base[.L]
BFINS:E:R.L	Rs[.L],	#offset,	#width,	base[.L]
BFINSU:G:I.W	#src[.BHWL],	offset[.BHWL],	width[.W],	base[.W]
BFINSU:G:R.W	Rs[.W],	offset[.BHWL],	width[.W],	base[.W]
BFINSU:G:I.L	#src[.BHWL],	offset[.BHWL],	width[.L],	base[.L]
BFINSU:G:R.L	Rs[.L],	offset[.BHWL],	width[.L],	base[.L]
BFINSU:E:I.W	#src[.BHWL],	#offset,	#width,	base[.W]
BFINSU:E:R.W	Rs[.W],	#offset,	#width,	base[.W]
BFINSU:E:I.L	#src[.BHWL],	#offset,	#width,	base[.L]
BFINSU:E:R.L	Rs[.L],	#offset,	#width,	base[.L]
BFCMP:G:I.W	#src[.BHWL],	offset[.BHWL],	width[.W],	base[.W]
BFCMP:G:R.W	Rs[.W],	offset[.BHWL],	width[.W],	base[.W]
BFCMP:G:I.L	#src[.BHWL],	offset[.BHWL],	width[.L],	base[.L]
BFCMP:G:R.L	Rs[.L],	offset[.BHWL],	width[.L],	base[.L]
BFCMP:E:I.W	#src[.BHWL],	#offset,	#width,	base[.W]
BFCMP:E:R.W	Rs[.W],	#offset,	#width,	base[.W]
BFCMP:E:I.L	#src[.BHWL],	#offset,	#width,	base[.L]
BFCMP:E:R.L	Rs[.L],	#offset,	#width,	base[.L]

【図149】

BFCMPU:G:I.W	#src[.BHWL],	offset[.BHWL],	width[.W],	base[.W]
BFCMPU:G:R.W	Rs[.W],	offset[.BHWL],	width[.W],	base[.W]
BFCMPU:G:I.L	#src[.BHWL],	offset[.BHWL],	width[.L],	base[.L]
BFCMPU:G:R.L	Rs[.L],	offset[.BHWL],	width[.L],	base[.L]
BFCMPU:E:I.W	#src[.BHWL],	#offset,	#width,	base[.W]
BFCMPU:E:R.W	Rs[.W],	#offset,	#width,	base[.W]
BFCMPU:E:I.L	#src[.BHWL],	#offset,	#width,	base[.L]
BFCMPU:E:R.L	Rs[.L],	#offset,	#width,	base[.L]
BFEXT:G.W	offset[.BHWL],	width[.W],	base[.W],	Rd[.W]
BFEXT:G.L	offset[.BHWL],	width[.L],	base[.L],	Rd[.L]
BFEXT:E.W	#offset,	#width,	base[.W],	Rd[.W]
BFEXT:E.L	#offset,	#width,	base[.L],	Rd[.L]
BFEXTU:G.W	offset[.BHWL],	width[.W],	base[.W],	Rd[.W]
BFEXTU:G.L	offset[.BHWL],	width[.L],	base[.L],	Rd[.L]
BFEXTU:E.W	#offset,	#width,	base[.W],	Rd[.W]
BFEXTU:E.L	#offset,	#width,	base[.L],	Rd[.L]

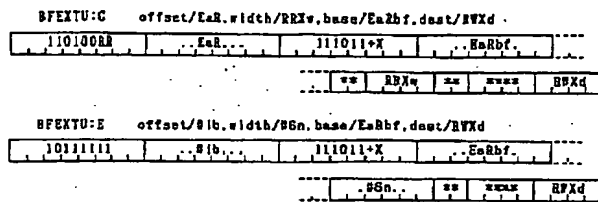
【図157】

命令	F	X	V	L	M	Z	コメント
BFINSU	-	-	+	-	+	+	

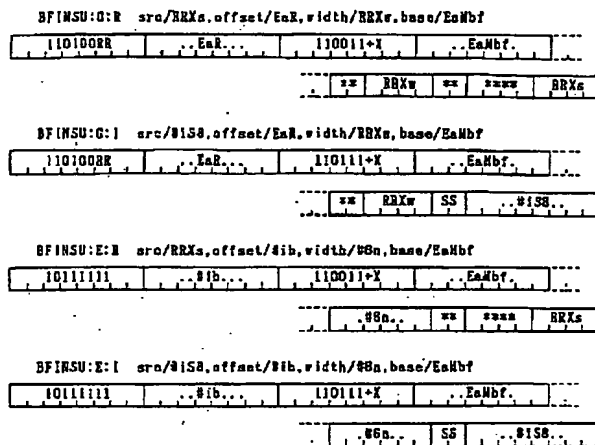
【図159】

命令	F	X	V	L	M	Z	コメント
BFCMP	-	-	-	+	-	+	

【図152】



【図156】



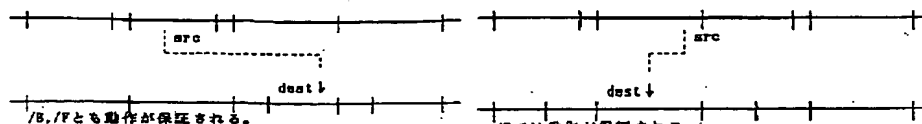
【図161】

命令	F	X	V	L	M	Z	コメント
BFCMPU	-	-	-	+	-	+	

【図163】

命令	F	X	V	L	M	Z	コメント
BVSCN	-	-	*	-	-	-	Vはサーチ失敗

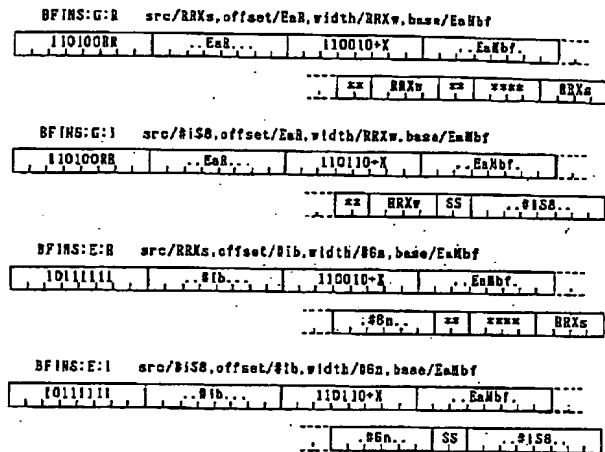
【図166】



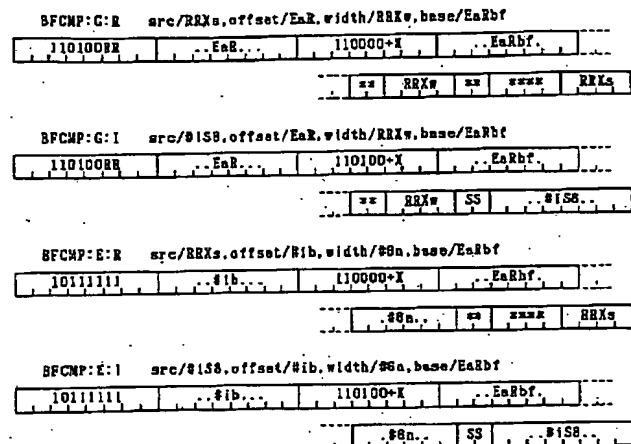
【図167】

/Fでは動作が保証される。
/Bでは結果が保証されず、ページフォールトの発生によって結果が変わることもある。

【図154】



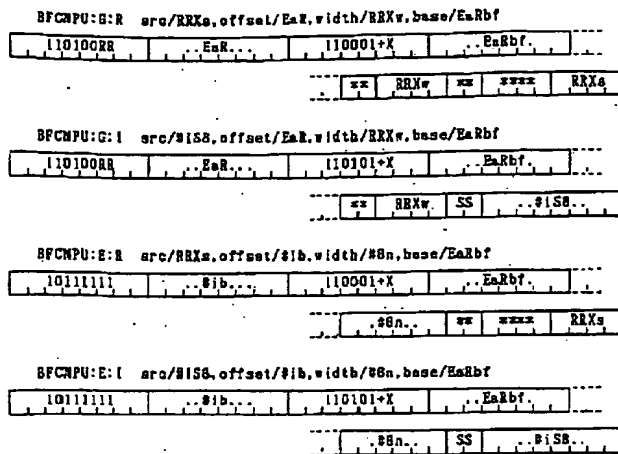
【図158】



【図165】

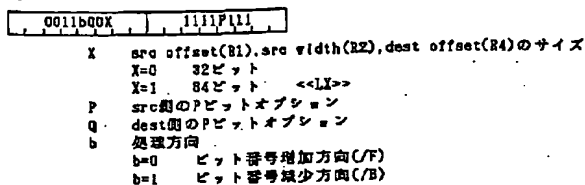
命令	F	X	V	L	M	Z	コメント
BVBNP	-	-	-	-	-	-	

【図160】



【図164】

BVMAP



レジスタのパラメータ

- R0: srcのビットフィールドのbase address
 R1: srcのビットフィールドのoffset
 符号付きとして扱われ、負の値も許される。
 R2: width
 演算するbitfieldの長さ(ビット数)
 width(R2)も符号付きとして扱われるが、width≤0の場合は、何もせずに命令を終了する。
 EITとはしない。
 R3: destのビットフィールドのbase address
 R4: destのビットフィールドのoffset
 符号付きとして扱われ、負の値も許される。
 R5: 演算の種類
 下位4bitを使用

【図170】

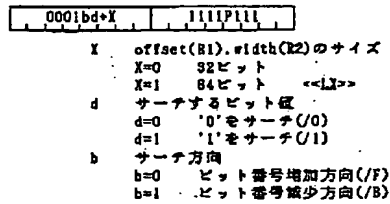
命令	F	X	V	L	M	Z	コメント
BVCPY	-	-	-	-	-	-	

【図172】

命令	F	X	V	L	M	Z	コメント
BVCPY	-	-	-	-	-	-	

【図162】

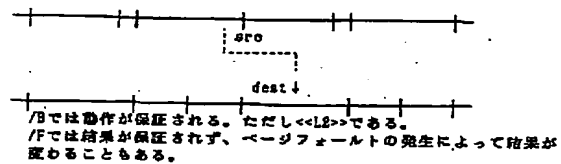
BVSCB



レジスタのパラメータ

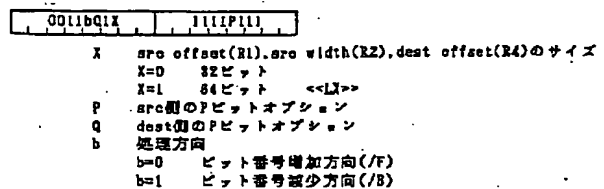
- R0: base address
 offset
 R1: read-modify-writeのオペランドになっており、パラメータである検索開始offsetとリターンパラメータである検索結果のoffsetが入る。dで指定した値のビットが見つかるまで、offsetは何回もワード境界を超える。命令中時には、その時点で検索中のoffsetが入る。
 R2: width
 offsetを検索するビットフィールドの長さ(ビット数)
 width(R2)も符号付きとして扱われるが、width≤0の場合は、Vflagのセットだけで行なってそのまま命令を終了する。
 EITとはしない。

【図168】



【図169】

BVCPY

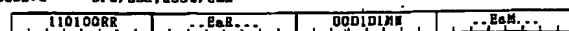


レジスタのパラメータ

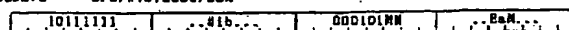
- R0: srcのビットフィールドのbase address
 R1: srcのビットフィールドのoffset
 符号付きとして扱われ、負の値も許される。
 R2: width
 演算するbitfieldの長さ(ビット数)
 width(R2)も符号付きとして扱われるが、width≤0の場合は、何もせずに命令を終了する。
 EITとはしない。
 R3: destのビットフィールドのbase address
 R4: destのビットフィールドのoffset
 符号付きとして扱われ、負の値も許される。

【図173】

ADDDX:G src/EaR,dest/EsM



ADDDX:E src/RSB,dest/EsM



【図171】

BVPAT

000001	X	1111P111
--------	---	----------

X pattern(R0), width(R2), dest offset(R4)のサイズ
 X=0 32ビット
 X=1 64ビット <<LX>>
 P dest側のPビットオプション

レジスタ上のパラメータ

R0: pattern
 R1: 使用しない
 R2: width
 演算するbitfieldの長さ(ビット数)
 width(R2)も符号付きとして扱われるが、width≤0の場合は、
 何もせずに命令を終了する。
 EITとはしない。
 R3: destのビットフィールドのbase address
 R4: destのビットフィールドのoffset
 符号付きとして扱われ、負の値も許される。
 R5: 演算の種類
 下位4bitを使用。BVPAT命令と共通。

【図174】

命令	F	X	V	L	M	Z	コメント
ADDDX	-	+	+	0	+	+	

【図177】

PACKss src/EaR,dest/EaR

110101RR	EaR	010010VV	EaR
----------	-----	----------	-----

【図175】

SUBDX:G src/EaR,dest/EaR

110100RR	EaR	000111MM	EaR
----------	-----	----------	-----

SUBDX:E src/R1b,dest/EaR

10111111	R1b	000111MM	EaR
----------	-----	----------	-----

【図176】

命令	F	X	V	L	M	Z	コメント
SUBDX	-	+	+	+	+	+	

【図179】

UNPKss src/EaR,dest/EaR,adj/R1V

110101RR	EaR	010011RR	EaR
			R1V

【図178】

命令	F	X	V	L	M	Z	コメント
PACKss	-	-	-	-	-	-	

【図180】

命令	F	X	V	L	M	Z	コメント
UNPKss	-	-	-	-	-	-	

【図182】

SNOV

00ssssSS	1110P1Qb
----------	----------

P src側のPビットオプション
 Q dest側のPビットオプション
 SS エレメントサイズ、終了条件(R3,R4)のサイズ
 b 処理方向
 b=0 アドレス増加の方向に処理する(/F)
 b=1 アドレス減少の方向に処理する(/B)
 ssss ストリング命令終了条件

【図188】

(フラグ変化)

命令	F	X	V	L	M	Z	コメント
SSCH	*	-	*	-	*	-	Vはサータ欠陥

レジスタ上のパラメータ

R0: src側ストリングの先頭アドレス
 R1: dest側ストリングの先頭アドレス
 R2: ストリングの長さ、データ数
 R3: 終了条件の比較値(1)
 R4: 終了条件の比較値(2) (ATONでは使用しない)

【図183】

命令	F	X	V	L	M	Z	コメント
SNOV	*	-	*	-	*	-	

【図185】

命令	F	X	V	L	M	Z	コメント
SCMP	*	*	*	+	*	+	X,L,Zは最終更新の比較

【図192】

命令	F	X	V	L	M	Z	コメント
QINS	-	-	-	-	-	*	

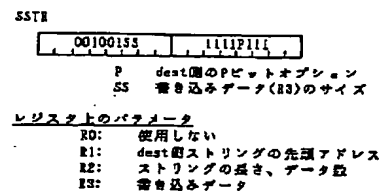
【図181】

```

UNPKBH  src(.B),dest(.H),adj(.H)
RR=00,WW=01    tmpは16ビット
                0 ==> tmp[00:15],
                src[00:03] ==> tmp[04:07],
                src[04:07] ==> tmp[12:15],
                tmp + adj ==> dest
UNPKHW  src(.H),dest(.W),adj(.W)
RR=01,WW=10    tmpは32ビット
                0 ==> tmp[00:31],
                src[00:03] ==> tmp[04:07],
                src[04:07] ==> tmp[12:15],
                src[08:11] ==> tmp[20:23],
                src[12:15] ==> tmp[28:31],
                tmp + adj ==> dest
UNPKBW  src(.B),dest(.W),adj(.W)
RR=00,WW=10    0 ==> tmp[00:31],
                src[00:03] ==> tmp[12:15],
                src[04:07] ==> tmp[28:31],
                tmp + adj ==> dest
UNPKWL  src(.W),dest(.L),adj(.L)
UNPKHL  src(.H),dest(.L),adj(.L)
                <<LX>>
                <<LX>>

```

【図189】



【図184】

SCMP

000000SS	1110PQQb
----------	----------

P src1側のPビットオプション
Q src2側のPビットオプション
SS エレメントサイズ、終了条件(R3,R4)のサイズ
b 処理方向
b=0 アドレス増加の方向に処理する(/F)
b=1 アドレス減少の方向に処理する(/B)
eeee ストリング命令終了条件

レジスタのパラメータ

R0: src1側ストリングの先頭アドレス
R1: src2側ストリングの先頭アドレス
R2: ストリングの長さ、データ数
R3: 終了条件の比較値(1)
R4: 終了条件の比較値(2) (ATOMでは使用しない)

【図186】

[SCMP終了要因]			V_flag	Z_flag	L_flag	F_flag	M_flag
長さ	不一致	終了条件			X_flag		
X	X	0(src1=src2)	0	1	0	1	*A
X	O	X(src1)	0	0	*C	0	0+
X	O	O(src1)	0	0	*C	1	*B
O	X	X	1	1	0	0	0+

【図190】

(フラグ設定)

命令	F	X	V	L	M	Z	コメント
SSR	-	-	-	-	-	-	

【図187】

SSCB

000000SS	1111P10r
----------	----------

P src側のPビットオプション
SS エレメントサイズ、終了条件(R3,R4)のサイズ
eeee ストリング命令終了条件
r ポインタ更新の方法
r=0 エレメントサイズだけ増加(/P)
r=1 増減値をレジスタR5で指定(/B)

レジスタのパラメータ

R0: src側ストリングの先頭アドレス
R1: 使用しない
R2: ストリングの長さ、データ数
R3: 終了条件の比較値(1)
R4: 終了条件の比較値(2) (ATOMでは使用しない)
R5: ポインタ更新値 (/Bオプションの時)

【図191】

QINS entry/EnRqP, queue/EnRqP2

11011000	EnRqP	101110+-	EnRqP2
----------	-------	----------	--------

【図196】

QDEL queue/EnRqP, dest/EnRq16

11011000	EnRqP	101100+-	EnRq16
----------	-------	----------	--------

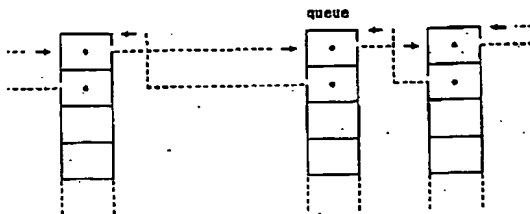
【図193】

```

address_of_queue ==> mem[address_of_entry] ==> temp1
mem[address_of_queue + 4] ==> mem[address_of_entry + 4] ==> temp2
address_of_entry ==> mem[mem[address_of_queue + 4]]
address_of_entry ==> mem[address_of_queue + 4]
if (temp1 = temp2) then
    1 ==> Z_flag
else
    0 ==> Z_flag
endif

```

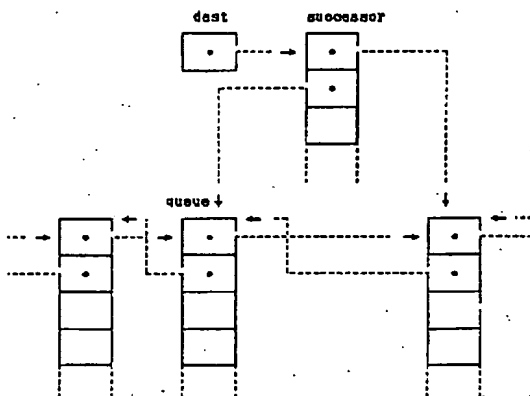
【図194】



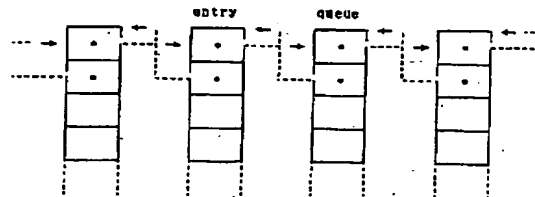
【図197】

命令	F	X	V	L	M	Z	コメント
QDEL	-	-	*	-	-	*	

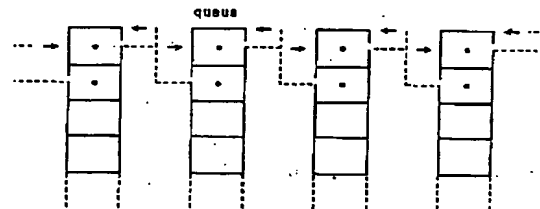
【図200】



【図195】



【図199】



【図201】

QSCR

00ccccSS	1111P0ab
----------	----------

- F キュー領域のPビットオプション
 SS 比較終了条件(R3, R4)とサーチデータのサイズ
 cccc 比較終了条件(ストリング命令終了条件と同じ)
 a マスクの有無
 a=0 RSのマスクなし(/NM)
 a=1 RSのマスクあり(/MR)
 b サーチ方向
 b=0 順方向(/F)
 b=1 逆方向(/B)

レジスタのパラメータ

- R0: サーチを開始するキューエントリのアドレス
 最初はQUEUE READの内容 = 最初のエントリのアドレスを入
 れる。
 R1: リターンパラメータとして使用される。命令終了時に、一つ
 前のキューエントリのアドレスが入っている。
 R2: キュー終了アドレス
 R3: 比較値(1)
 R4: 比較値(2)
 R5: エントリ中のサーチデータのオフセット
 サーチするメンバのリンクアドレスからのオフセット
 R6: マスク (X=1の時)

【図198】

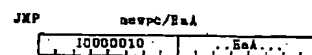
```

mem[address_of_queue] ==> successor
if(address_of_queue successor) then
    1 ==> V_flag
    1 ==> Z_flag
else
    successor ==> dest
    mem[successor] ==> mem[address_of_queue] ==> temp1
    address_of_queue ==> mem[mem[successor] + 4] ==> temp2

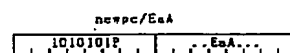
    if (temp1 = temp2) then
        0 ==> V_flag
        1 ==> Z_flag
    else
        0 ==> V_flag
        0 ==> Z_flag
    endif
endif

```

【図212】



【図214】

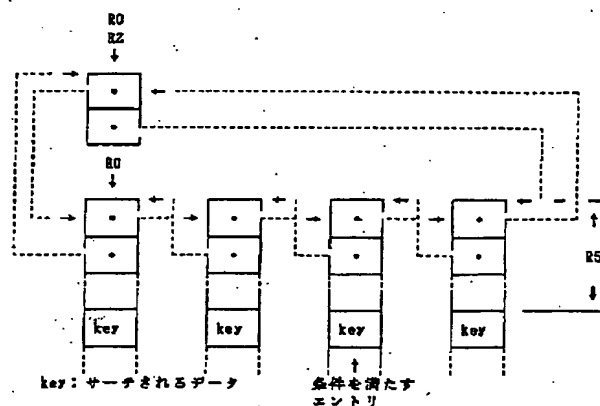
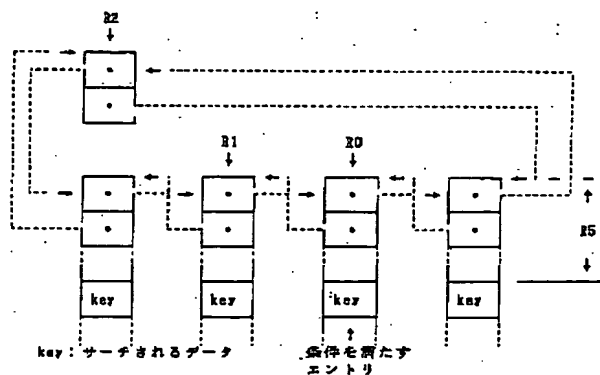


【図202】

命令	F	X	V	L	M	Z	コメント
QSCN	*	-	*	-	*	-	Vはサーチ失敗

【図204】

【図205】

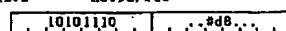


【図206】

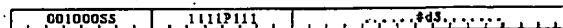
【図207】

命令	F	X	V	L	M	Z	コメント
BRA	-	-	-	-	-	-	

BRA:D newpc/8d8



BRA:U newpc/8d5



【図211】

命令	F	X	V	L	M	Z	コメント
BSR	-	-	-	-	-	-	

【図209】

命令	F	X	V	L	M	Z	コメント
Bcc	-	-	-	-	-	-	

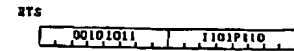
【図203】

```

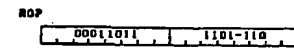
while (1) do
  RO ==> R1
  if b=0 then
    mem[RO] ==> RO    /*順方向のリンクをたどる*/
  else
    mem[RO+4] ==> RO  /*逆方向のリンクをたどる*/
  if(RO = R2) then
    1 ==> V_flag
    0 ==> M_flag
    0 ==> F_flag
    exit              /*サーチ失敗*/
  endif
  if m=0 then
    compare mem[RO+R5] with R3, R4
    and set F_flag, M_flag according to eeee
    /*終了条件成立の場合F_flag=1となる*/
  else
    compare (mem[RO+R5] & R6) with R3, R4
    and set F_flag, M_flag according to eeee
    /*終了条件成立の場合F_flag=1となる*/
  endif
  if (F_flag = 1) then
    0 ==> V_flag
    exit              /*サーチ成功*/
  endif
  check_interrupt
end_while

```

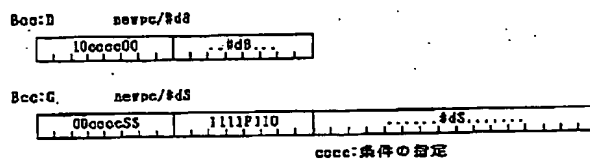
【図227】



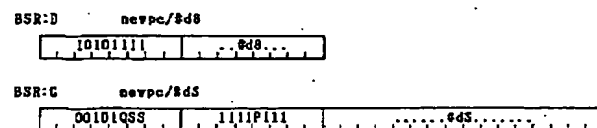
【図229】



【図208】



【図210】



【図213】

命令	F	X	V	L	M	Z	コメント
JMP	-	-	-	-	-	-	

【図215】

命令	F	X	V	L	M	Z	コメント
JSR	-	-	-	-	-	-	

【図217】

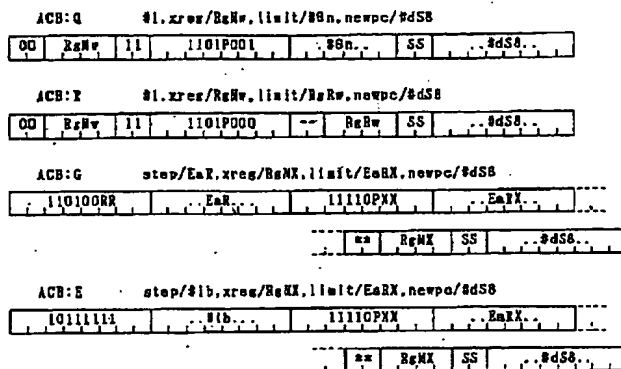
命令	F	X	V	L	M	Z	コメント
ACB	-	-	-	-	-	-	

【図218】

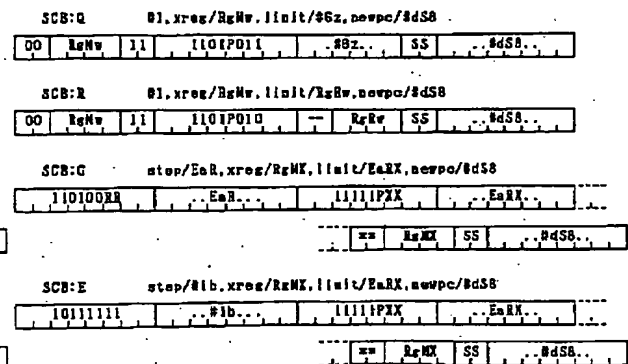
ACB:0 00R#11 1101P001 #8# SS #d5#
 00000011 11010001 00010001 00000000 00010010 00110100

<アドレス> +0 +1 +2 +3 +4 +5

【図216】



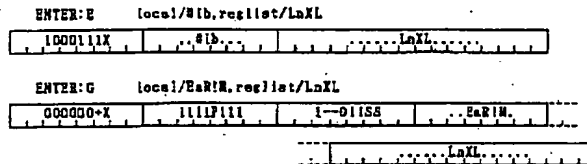
【図219】



【図220】

命令	F	X	V	L	M	Z	コメント
SCB	-	-	-	-	-	-	

【図221】

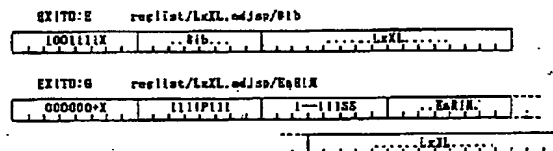


【図222】

命令	F	X	V	L	M	Z	コメント
ENTER	-	-	-	-	-	-	

【図223】

【図224】



【図225】

命令	F	X	V	L	M	Z	コメント
EXITD	-	-	-	-	-	-	

【図226】

【図228】

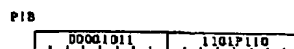


命令	F	X	V	L	M	Z	コメント
NSB	-	-	-	-	-	-	

【図230】

【図231】

命令	F	X	V	L	M	Z	コメント
NDP	-	-	-	-	-	-	



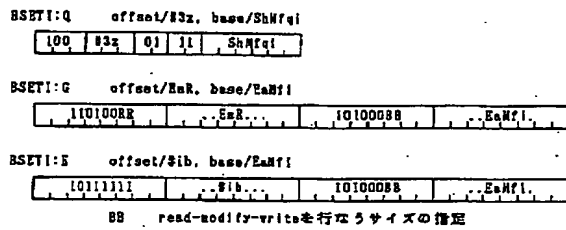
【図232】

【図234】

命令	F	X	V	L	M	Z	コメント
PIB	-	-	-	-	-	-	

命令	F	X	V	L	M	Z	コメント
BSPT	-	-	-	-	-	-	Zがテスト結果

【図233】



【図236】

命令	F	X	V	L	M	Z	コメント
BCLRl	-	-	-	-	-	+	Zがテスト結果

【図238】

命令	F	X	V	L	M	Z	コメント
CS	-	-	-	-	-	+	Zは更新成功

【図240】

命令	F	X	V	L	M	Z	コメント
LDC	-	-	-	-	-	-	-destがPSVの場合

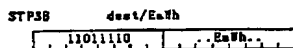
【図242】

命令	F	X	V	L	M	Z	コメント
STC	-	-	-	-	-	-	

【図244】

命令	F	X	V	L	M	Z	コメント
LDPSh	*	*	*	*	*	*	命令により設定

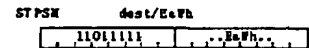
【図247】



命令	F	X	V	L	M	Z	コメント
STPSB	-	-	-	-	-	-	

【図248】

【図249】



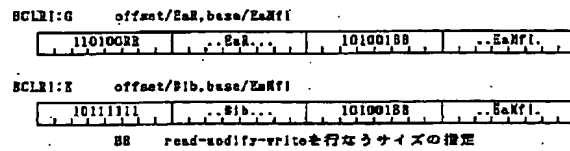
【図250】

命令	F	X	V	L	M	Z	コメント
STPSH	-	-	-	-	-	-	

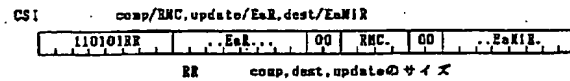
【図252】

命令	F	X	V	L	M	Z	コメント
LDP	-	-	-	-	-	-	

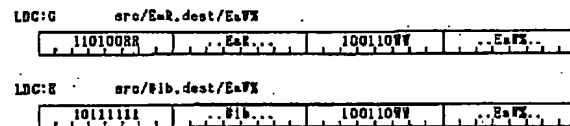
【図235】



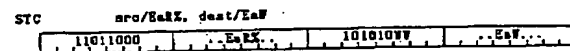
【図237】



【図239】

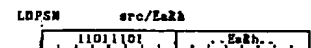
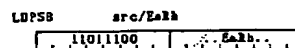


【図241】



【図243】

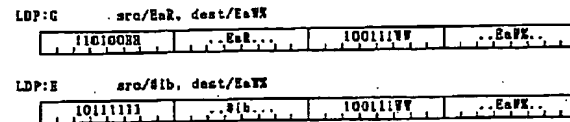
【図245】



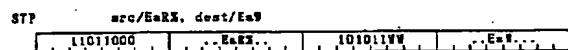
【図246】

命令	F	X	V	L	M	Z	コメント
LDPSh	-	-	-	-	-	-	

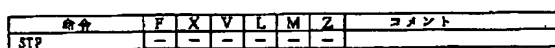
【図251】



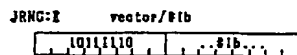
【図253】



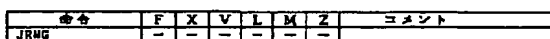
【図254】



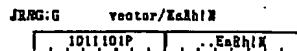
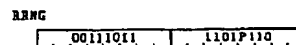
【図255】



【図256】

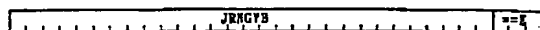


【図263】



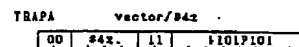
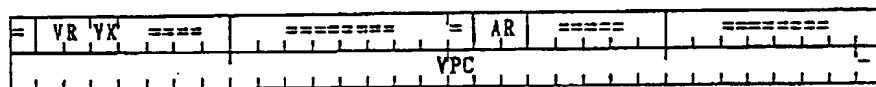
【図264】

【図257】

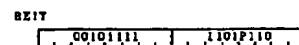


【図268】

【図258】



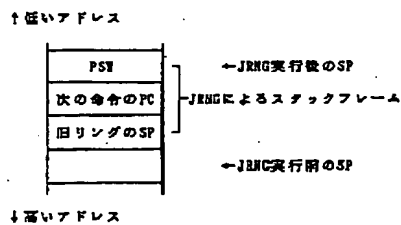
【図272】



VR(Vector RNG):JRNG命令実行によって新しく移行する先のリング番号
 AR(Access RNG):JRNG命令の実行が許可される最も外側のリング番号
 VX:(Vector XA):JRNG命令実行後の新XA
 現在は0に固定
 VPC(Vector PC):JRNG命令実行後の新PC

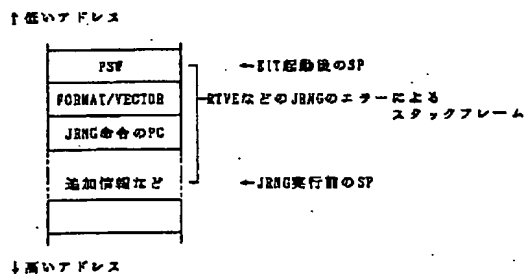
【図259】

JRNGで形成されるスタックフレーム(新リング)



【図260】

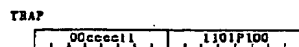
(A)JRNGでEITが発生した場合のスタックフレーム(正)



【図269】



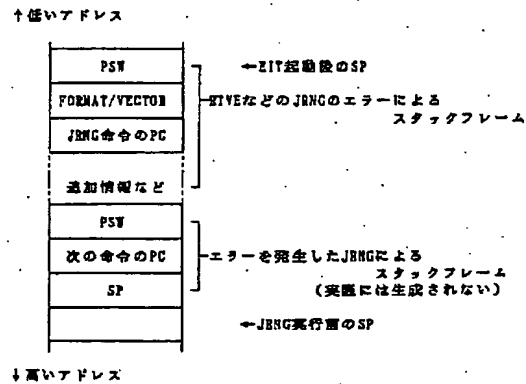
【図270】



ccccは条件指定

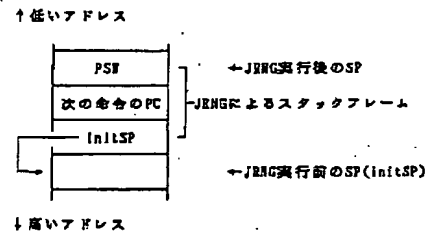
【図261】

(B) JRNNGでEITが発生した場合のスタックフレーム (簡)

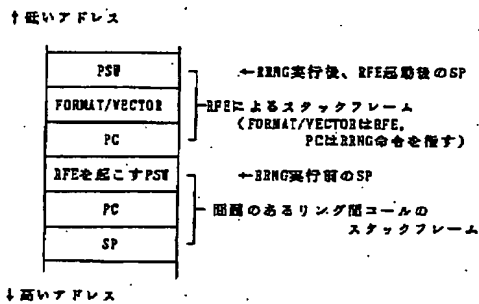


【図262】

JRNNGで同じリングにジャンプする場合のスタックフレーム

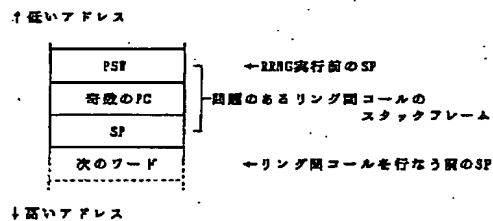


【図265】



※リング関コールを行なった側のSPは元のままである。
 ※PSWは、RNG命令実行前のPSWの値を、RFE、RIVEのEITVTEによって書き換えたものになる。
 PRNGは、RNG命令を実行したリングを指す。スタック中に返還されていたPSW (RFEを起こすPSW) は、EIT起動後のPSWには影響しない。

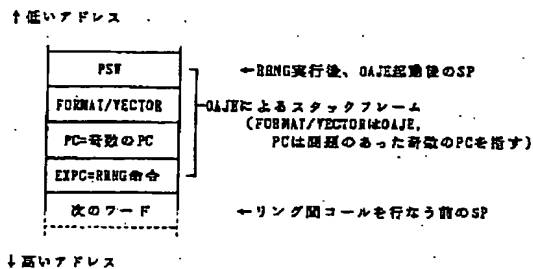
【図266】



【図273】

命令	F	X	V	L	M	Z	コメント
REIT	*	*	*	*	*	*	スタックより復帰

【図267】



※リング関コールを行なった側のSPは、スタックから復帰されたものになる。
 ※PSWは、一旦スタックから復帰した値を、OAJEのEITVTEによって書き換えたものになる。
 PRNGは、リング関コールを行なったリングを指す。つまり、ソフトウェアによってPRNGを書き換えていない限り、RNG命令実行前のPRNGの値と同じである。

【図271】

命令	F	X	V	L	M	Z	コメント
TRAP	-	-	-	-	-	-	

【図275】

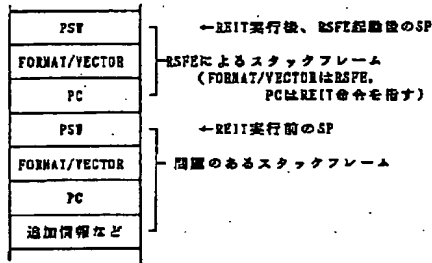
命令	16mask/01h
WAIT	00001111 1701-11001h.....

【図276】

命令	F	X	V	L	M	Z	コメント
WAIT	-	-	-	-	-	-	

【図274】

↑低いアドレス



【図278】

命令	F	X	V	L	M	Z	コメント
LDCTX	-	-	-	-	-	-	

【図280】

命令	F	X	V	L	M	Z	コメント
STCTX	-	-	-	-	-	-	

【図283】

命令	F	X	V	L	M	Z	コメント
MOVPA	-	-	-	-	-	-	

レジスタのパラメータ
R1 アドレス変換テーブルのベースアドレス

【図285】

	V_flag	F_flag	結果
正常終了	0	0	物理アドレス ==> dest
エラー	1	0	destは変化しない
ページファクト(ST,PT,PAGE)	1	1	destは変化しない

エラーには、ATEのフォーマットエラー（予約ATEエラー）や、ATEによって未使用領域の指定があった場合が含まれる。

【図287】

命令	F	X	V	L	M	Z	コメント
LDATE	-	-	-	-	-	-	

LDATE sro/EnR,destaddr/EnA

11010118	EnA...	10ptt00	EnA...
----------	--------	---------	--------

p ページを行なう論理空間の指定
p=0 すべての空間(/LS)
p=1 ROで指定したLSIDを持つ空間(/SS)

ttt ロードを行なうATEの指定
ttt=000 PTEへのロード
ttt=001 STEへのロード

レジスタのパラメータ
RO ページを行なうTLBの論理空間のLSID (/SSの時のみ)
R1 アドレス変換テーブルのベースアドレス

【図277】

命令	F	X	V	L	M	Z	コメント
LDCTX	-	-	-	-	-	-	

LDCTX ctxaddr/EnA!A

10xx0110	EnA...
----------	--------

xx CTXBを置く空間の指定
xx=00 論理空間 (/LS)
xx=01 物理空間 (/CS)
xx=10 reserved
xx=11 reserved

【図281】

命令	F	X	V	L	M	Z	コメント
ACS	-	-	-	-	-	-	

ACS chkaddr/ZsA

10000011	EnA...
----------	--------

【図279】

命令	F	X	V	L	M	Z	コメント
STCTX	-	-	-	-	-	-	

STCTX

00xx0111	1101P110
----------	----------

xx CTXBを置く空間の指定
xx=00 論理空間 (/LS)
xx=01 物理空間 (/CS)
xx=10 reserved
xx=11 reserved

【図282】

命令	F	X	V	L	M	Z	コメント
ACS	-	-	-	-	-	-	

【図284】

命令	F	X	V	L	M	Z	コメント
MOVPA	-	-	-	-	-	-	

【図286】

アドレス	一般のメモリアクセス	MOVPA,LDATE,STATE命令
H'00000000-7fffffff	UATBによりアドレス変換	R1によりアドレス変換
H'80000000-ffffffff	SATBによりアドレス変換	SATBによりアドレス変換

【図288】

命令	F	X	V	L	M	Z	コメント
LDATE	-	-	-	-	-	-	

【図289】

	V_flag	F_flag	結果
正常終了	0	0	ATEはセットされる
ロードしたATEのPI=0	0	1	ATEはセットされる
ロードしたATEの予約ATEエラー	0	1	ATEはセットされる
途中段ATEの予約ATEエラー	1	0	ATEはセットされない
途中段ATEのPI=0 (ページ不在)	1	1	ATEはセットされない

V_flagは予約ATEエラーやページ不在によって、ATEへの設定ができなかったことを示す

【図290】

命令	F	X	V	L	M	Z	コメント
STATE	0	0	0	0	0	0	

【図291】

STATE	srcaddr/EaA, dest/EaBIS
11011000	...EaA... 100ttt+8 ...EaBIS...
ttt	ストアを行なうATEの指定 ttt=000 PTEからのストア ttt=001 STEからのストア
レジスタのパラメータ	
R1	アドレス変換テーブルのベースアドレス

【図292】

	V_flag	F_flag	結果
正常終了	0	0	ATE ==> dest
読み出されたATEのPI=0	0	1	ATE ==> dest
読み出されたATEの予約ATEエラー	0	1	ATE ==> dest
途中段のATEの予約ATEエラー	1	0	destは変化しない
途中段のページアクト	1	1	destは変化しない

V_flagは、予約ATEエラーやページアクトによって、ATEの読み出しができなかったことを示す。
なお、予約ATEエラーは、ATEのreservedのビットを使用していた場合などに発生するものである。

【図293】

PTLB	
00p11111	1101P110
p	ページを行なう論理空間の指定 p=0 すべての空間(/AS) p=1 R0で指定したLS10を持つ空間(/SS)
レジスタのパラメータ	
R0	ページを行なうTLBの論理空間のLS10 (/SSの時のみ)

【図294】

命令	F	X	V	L	M	Z	コメント
PTLB	0	0	0	0	0	0	

【図295】

PSTLB	srcaddr/EaA
00000000	1111P111 0-pttt...EaA...
p	ページを行なう論理空間の指定 p=0 すべての空間(/AS) p=1 R0で指定したLS10を持つ空間(/SS)
ttt	ページを行なう論理アドレス範囲の指定 ttt=000 論理アドレス全体(2 ³¹ ~2 ¹² ビット)が一致するエントリをページ(/PT) ttt=001 論理アドレスの2 ³¹ ~2 ²² ビットが一致するエントリをページ(/ST) ttt=110 物理アドレスの2 ³¹ ビットが一致するエントリをページ(/AT)
レジスタのパラメータ	
R0	ページを行なうTLBの論理空間のLS10 (/SSの時のみ)

【図296】

命令	F	X	V	L	M	Z	コメント
PSTLB	0	0	0	0	0	0	

【図297】

AT	意味
00	アドレス変換、メモリ保護なし
01	reserved
10	アドレス変換なし、アドレスのみを使った簡単なメモリ保護 <<L1B>> (アドレスのMSBによるメモリ領域の区別、2リング)
11	reserved

【図301】

bit0	25	26	27	28	29	30	31
	STB	-	-	0	0	0	PI

各フィールドのビットは SAT0 と同じである。

【図308】

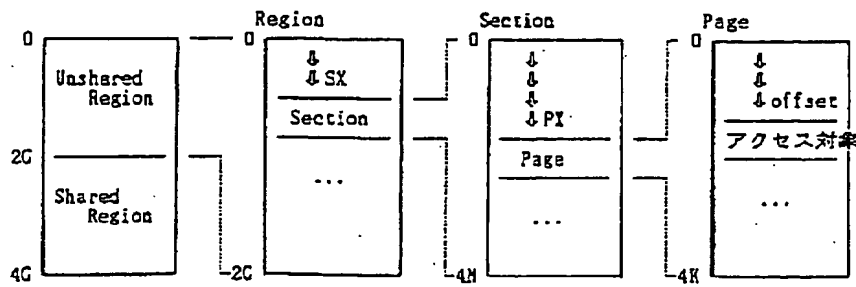
【図298】

bit0	1	9	10	19	20	31
R	SX		PX		offset	
R:	領域(Region)指定子				1ビット	
SX:	Section index				9ビット	
PX:	Page index				10ビット	
offset:	Page offset				12ビット	

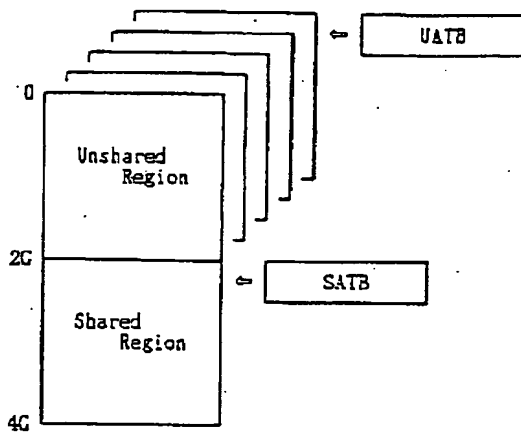
D	LL	使用できるSXの値の範囲(2進ビット対応)
0	00	00000000 512 エントリ
0	01	00000000 256 エントリ
0	10	00000000 64 エントリ
0	11	00000000 16 エントリ
1	00	(別項参照)
1	01	11000000 256 エントリ
1	10	11110000 64 エントリ
1	11	11111111 16 エントリ

* 12 0/1 の両方の値をとれることを示す

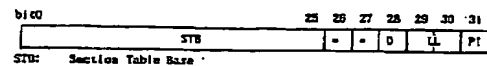
【図299】



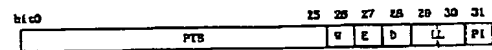
【図300】



【図302】



【図307】



【図303】

D	LL	使用できるSXの値の範囲(2進ビット対応)	
=	00	* * * * *	512 エントリ
0	01	0 * * * * *	256 エントリ
0	10	0 0 0 * * * *	64 エントリ
0	11	0 0 0 0 0 * * *	16 エントリ
1	01	1 * * * * *	256 エントリ
1	10	1 1 1 * * * *	64 エントリ
1	11	1 1 1 1 1 * * *	16 エントリ

* は 0/1 の両方の値をとれることを示す

= は '0' に reserved であるが、インプリメント上は違反時も無視することを示す。

【図304】

(1/2 (最大)サイズ の Section Table = 2KB)

STB(D=0, LL=00)→

SX=B'000000000 の STE
SX=B'000000001 の STE
SX=B'000001111 の STE
SX=B'000010000 の STE
SX=B'111101111 の STE
SX=B'111110000 の STE
SX=B'111111110 の STE
SX=B'111111111 の STE

| : 使用する領域

| : 未使用領域

〈例①〉 〈例②〉

【図311】

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PF	PL	PC	AL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL	IL

【図305】

(1/64 サイズ の Section Table = 64B)

STB(D=0, LL=11)→

SX=B'000000000 の STE
SX=B'000000001 の STE
SX=B'000001111 の STE
これ以上のアドレス値を 指定すると AIRE

| : 使用する領域

| : 未使用領域

〈例①〉

【図309】

D	LL	WE	動作
1	00	0 *	アドレス変換例外(AIRE) の 未使用領域参照エラーを発生
1	00	1 *	(reserved) アドレス変換例外(AIRE) の 予約ATEエラーを発生

【図 306】

[1/64 サイズの Section Table = 64B]

1:使用する領域

! : 未使用領域

〈例②〉

STB(D=1,LL=11)→	SX=B' 000000000 相当のSTE
	SX=B' 000000001 相当のSTE
	これ以下のアドレス値を 指定すると ATRE
	SX=B' 111110000 の STE
	SX=B' 111111110 の STE
	SX=B' 111111111 の STE

【図 3 1 0】

bit0	10	20	32
R(=SX==)<==PX==>(==offset==)			ATB, STE の LL=00
R(=SX==)<==PX==>(==offset==)			ATB, STE の LL=01
R(=SX==)<==PX==>(==offset==)			ATB, STE の LL=10
R(=SX==)<==PX==>(==offset==)			ATB, STE の LL=11

-: アドレスが 0 または 1 に固定される範囲

【图 3 2 3】

```

                                1 ← 挿入されるビットフィールド
src:                SSSSSSSSSSSSSSSSSSS
base+offset: 00000000000000000000000000000000
                                |offset|width|
                                base

```

【図 3 1 3】

T	AL	RL	動作
00	00	??	RL は本来の意味で使用
00	01	??	アドレス変換例外(ATE) の 未使用領域参照エラーを発生
00	10	??	(reserved) アドレス変換例外(ATE) の 予約ATEエラーを発生
00	11	??	(reserved) アドレス変換例外(ATE) の 予約ATEエラーを発生

* はソフトウェアで自由に使用してよいビットである。ハードウェア的にはこのビットが無視される。

【図312】

AL	RL	T=00				T=01				T=10				T=11			
		RO	R1	R2	R3	RO	R1	R2	R3	RO	R1	R2	R3	RO	R1	R2	R3
00	00	R	—	—	—	R-E	—	—	—	RW	—	—	—	RWE	—	—	—
00	01	R	R	—	—	R-E	R-E	—	—	RW	R	—	—	RWE	R-E	—	—
00	10	R	R	R	—	R-E	R-E	R-E	—	RW	R	R	—	RWE	R-E	R-E	—
00	11	R	R	R	R	R-E	R-E	R-E	R-E	RW	R	R	R	RWE	R-E	R-E	R-E
01	00	別表参照				R-E	—E	—	—	RW	—	—	—	RWE	—E	—	—
01	01					R-E	R-E	—	—	RW	RW	—	—	RWE	RWE	—	—
01	10					R-E	R-E	R-E	—	RW	RW	R	—	RWE	RWE	R-E	—
01	11					R-E	R-E	R-E	R-E	RW	RW	R	R	RWE	RWE	R-E	R-E
10	00	別表参照				R-E	—E	—E	—	RW	—	—	—	RWE	—E	—E	—
10	01					R-E	R-E	—E	—	RW	RW	—	—	RWE	RWE	—E	—
10	10					R-E	R-E	R-E	—	RW	RW	RW	—	RWE	RWE	RWE	—
10	11					R-E	R-E	R-E	R-E	RW	RW	RW	R	RWE	RWE	RWE	R-E
11	00	別表参照				R-E	—E	—E	—E	RW	—	—	—	RWE	—E	—E	—E
11	01					R-E	R-E	—E	—E	RW	RW	—	—	RWE	RWE	—E	—E
11	10					R-E	R-E	R-E	—E	RW	RW	RW	—	RWE	RWE	RWE	—E
11	11					R-E	R-E	R-E	R-E	RW	RW	RW	RW	RWE	RWE	RWE	RWE

【図357】

FB	—	—	RG	SP	MM
----	---	---	----	----	----

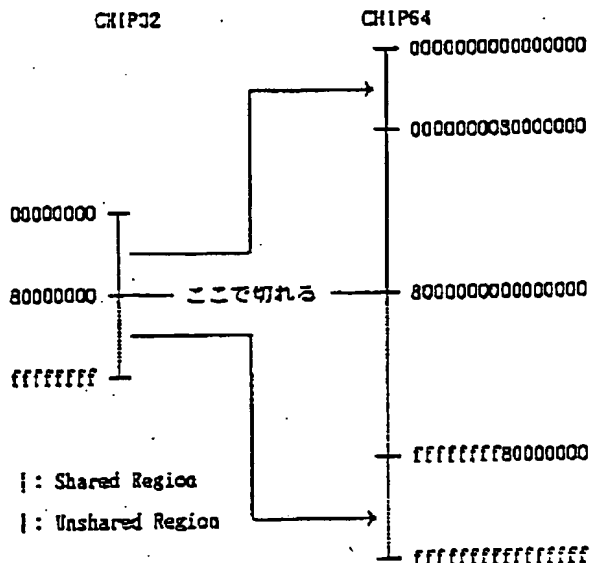
【図360】

PSW			
Format	Type	0000000	Vector
PC			
その他の情報			

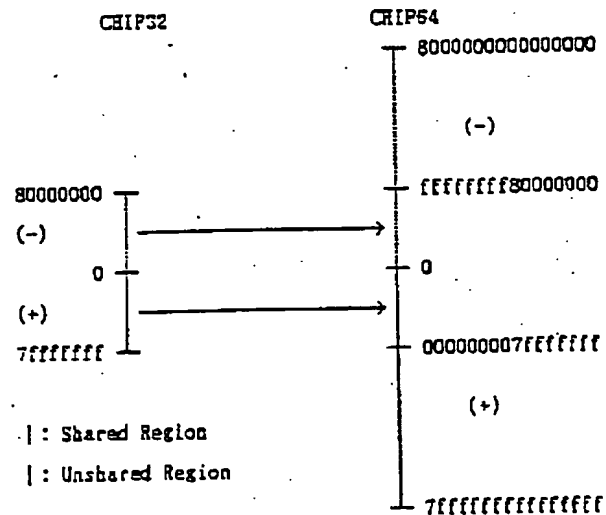
【図416】

$$\begin{array}{l}
 10000000(2) = -128(10) \\
 +) 10000000(2) = -128(10) \\
 \hline
 10000000(2) \quad -256 \\
 \text{↑ 最上位ビット}
 \end{array}$$

【図314】



【図315】



【図316】

命令	F	X	V	L	M	Z	コメント
CMP	—	—	—	+	—	+	
CMPI	—	—	—	+	—	+	
CHK	—	—	*	+	—	+	L, Zは下位値との比較

【図317】

命令	F	X	V	L	M	Z	コメント
MOV	-	-	+	-	+	+	
MOVU	-	-	+	-	+	+	
PUSH	-	-	-	-	-	-	
POP	-	-	-	-	-	-	
STN	-	-	-	-	-	-	
LDN	-	-	-	-	-	-	
NOVA	-	-	-	-	-	-	
PUSHA	-	-	-	-	-	-	

【図319】

命令	F	X	V	L	M	Z	コメント
AND	-	-	-	-	+	+	
OR	-	-	-	-	+	+	
XOR	-	-	-	-	+	+	
NOT	-	-	-	-	+	+	

【図320】

命令	F	X	V	L	M	Z	コメント
SHL	-	+	-	-	+	+	
SHA	-	+	+	-	+	+	
ROT	-	+	-	-	+	+	
SHXL	-	+	-	-	+	+	
SHXB	-	+	-	-	+	+	
RYBY	-	-	-	-	-	-	
RYBI	-	-	-	-	-	-	

【図322】

命令	F	X	V	L	M	Z	コメント
BFEKT	-	-	+	-	+	+	
BFEKTU	-	-	+	-	+	+	
BFINS	-	-	+	-	+	+	
BFINSU	-	-	+	-	+	+	
BFCMP	-	-	-	+	-	+	
BFCMPU	-	-	-	+	-	+	

【図325】

命令	F	X	V	L	M	Z	コメント
ADDX	-	+	+	0	+	+	
SU30X	-	+	+	+	+	+	
PACK ₃₂	-	-	-	-	-	-	
UNPK ₃₂	-	-	-	-	-	-	

【図327】

命令	F	X	V	L	M	Z	コメント
QINS	-	-	-	-	-	*	
QDEL	-	-	*	-	*	-	
QSCH	*	-	*	-	*	-	Vはサーチ失敗

【図318】

命令	F	X	V	L	M	Z	コメント
ADD	-	+	+	+	+	+	
ADDU	-	+	+	0	+	+	
ADDX	-	+	+	+	+	+	
SUB	-	+	+	+	+	+	
SUBU	-	+	+	+	+	+	
SUBX	-	+	+	+	+	+	
MUL	-	+	+	+	+	+	
MULU	-	+	0	+	+	+	
MULX	*	-	0	0	+	+	M,Zはdest基準、Fはtrap=0
DIV	-	-	1	0	+	0	-(最小負数)+(-1)の場合 -0除算, EITの場合
DIVU	-	-	1	0	+	+	-0除算, EITの場合
DIVX	*	-	1	0	+	+	M,Zはdest基準、Fはtrap=0 -dest=A'-7B-の場合 -0除算, EITの場合
REM	-	-	1	0	+	+	-0除算, EITの場合
REMU	-	-	0	0	+	+	-0除算, EITの場合
NEG	-	-	0	0	+	+	
INDEX	-	-	+	+	+	+	M,ZはXreg基準

【図321】

命令	F	X	V	L	M	Z	コメント
BIST	-	-	-	-	-	+	Zがテスト結果
BSET	-	-	-	-	-	+	Zがテスト結果
BCLB	-	-	-	-	-	+	Zがテスト結果
BNOT	-	-	-	-	-	+	Zがテスト結果
BSCH	-	-	*	-	-	-	Vはサーチ失敗

【図324】

命令	F	X	V	L	M	Z	コメント
BVSCB	-	-	*	-	-	-	Vはサーチ失敗
BVNAP	-	-	-	-	-	-	
BVCPU	-	-	-	-	-	-	
BVPAT	-	-	-	-	-	-	

【図326】

命令	F	X	V	L	M	Z	コメント
SNOV	*	-	*	-	*	-	
SCMP	*	*	*	+	*	+	X,L,Zは最終要素の比較
SSCH	-	-	*	-	*	-	
SSTB	-	-	-	-	-	-	Vはサーチ失敗

【図329】

命令	F	X	V	L	M	Z	コメント
BSETI	-	-	-	-	-	+	Zがテスト結果
BCLBI	-	-	-	-	-	+	Zがテスト結果
CSI	-	-	-	-	-	+	Zは更新成功

【図328】

命令	F	X	V	L	M	Z	コメント
BRA	-	-	-	-	-	-	
Bcc	-	-	-	-	-	-	
BSR	-	-	-	-	-	-	
JMP	-	-	-	-	-	-	
JSR	-	-	-	-	-	-	
ACB	-	-	-	-	-	-	
SCB	-	-	-	-	-	-	
ENTER	-	-	-	-	-	-	
EXITD	-	-	-	-	-	-	
RTS	-	-	-	-	-	-	
RDP	-	-	-	-	-	-	
PIB	-	-	-	-	-	-	

【図330】

命令	F	X	V	L	M	Z	コメント
LDC	-	-	-	-	-	-	-destがPSTの場合
STC	*	*	*	*	*	*	命令により設定
LDPSP	*	*	*	*	*	*	
LDPSM	*	*	*	*	*	*	
STPSP	*	*	*	*	*	*	
STPSM	*	*	*	*	*	*	
LDP	-	-	-	-	-	-	
STP	-	-	-	-	-	-	

【図332】

【図331】

命令	F	X	V	L	M	Z	コメント
JRNG	-	-	-	-	-	-	
RBRG	*	*	*	*	*	*	スタックより復帰
TRAPA	*	*	*	*	*	*	
TRAP	*	*	*	*	*	*	スタックより復帰
REIT	*	*	*	*	*	*	
WAIT	-	-	-	-	-	-	
LOGTX	-	-	-	-	-	-	
STGTX	-	-	-	-	-	-	

命令	F	X	V	L	M	Z	コメント
ACS	-	-	-	*	*	*	
MOVPA	*	*	*	*	*	*	
LDATE	*	*	*	*	*	*	
STATE	*	*	*	*	*	*	
PTLB	-	-	-	-	-	-	
PSTLB	-	-	-	-	-	-	
PLCH	-	-	-	-	-	-	
PSLCH	-	-	-	-	-	-	

【図334】

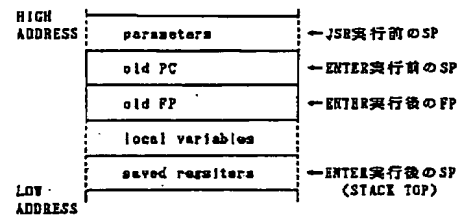
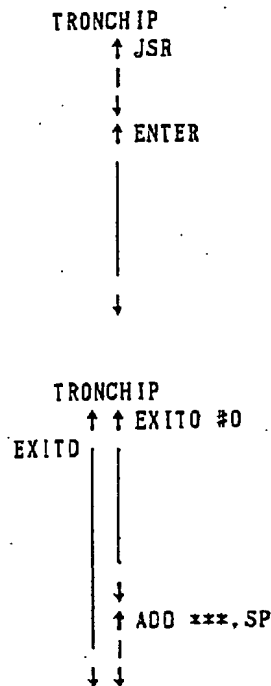
【図333】

<サブルーチンの入口>

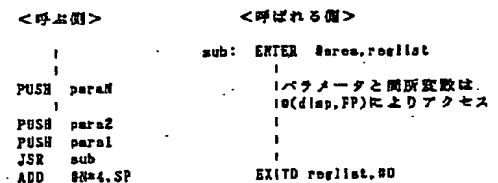
1. PCの退避と新しいPCの設定
2. FPの退避と新しいFPの設定
3. ローカル変数の領域の確保
4. レジスタの退避

<サブルーチンの出口>

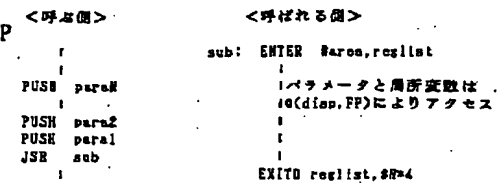
5. レジスタの復帰
6. ローカル変数解放とFPの復帰
7. PCの復帰とリターン
8. スタック上のパラメータ解放



【図335】



【図336】



【図341】

XXXXXXXX	XXXX	INASR	XXXXXXXXXX	XXXXXXXXXX
----------	------	-------	------------	------------

【図337】

```

procedure proc0 {レキシカルレベル0};
var var0;
procedure proc1A {レキシカルレベル1};
var var1A;
  procedure proc2A {レキシカルレベル2};
  var var2A;
  begin
    ...
  end
  procedure proc2B {レキシカルレベル2};
  var var2B;
  begin
    ...
  end
  begin {procedure proc1A}
    ...
  end
end
procedure proc1B {レキシカルレベル1};
var var1B;
  procedure proc2C {レキシカルレベル2};
  var var2C;
  begin
    ...
  end
  procedure proc2D {レキシカルレベル2};
  var var2D;
  begin
    ...
  end
  begin {procedure proc1B}
    ...
  end
begin {procedure proc0}
  ...
end

```

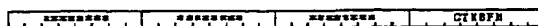
【図340】



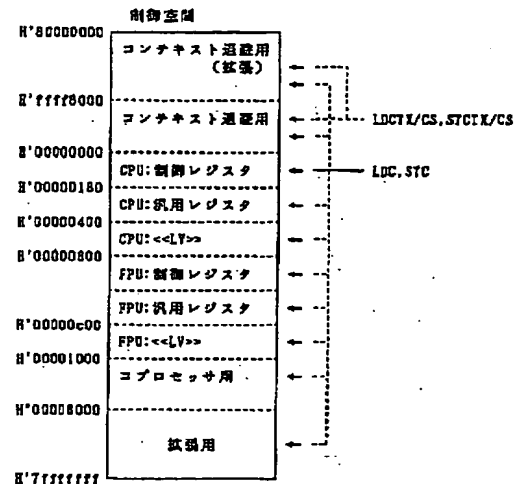
【図344】



【図347】



【図339】



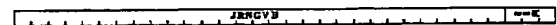
【図343】



【図346】



【図349】



【図342】

【図345】

【図348】

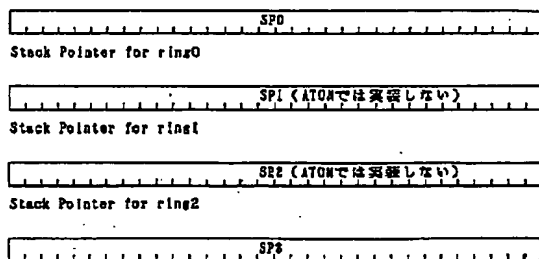
【図338】

<<サブルーチンコール状態>>

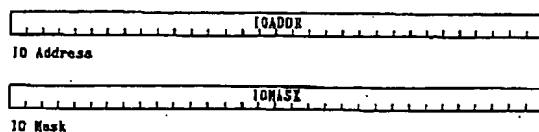
<<ディスプレイ>>

	FP lev=X	R13 lev=0	R12 lev=1	R11 lev=2	R10 lev=3
proc0	proc0 (var0)	proc0	-	-	-
↓					
proc1A [lev. up, FP -> R12]	proc1A (var1A)	proc0	proc1A		
↓					
proc2B [lev. up, FP -> R11]	proc2B (var2B)	proc0	proc1A	proc2B	
↓					
proc2A [lev. same, FP -> R11]	proc1A (var2A)	proc0	proc1A	proc2A	
↓					
proc0(recursion) [lev. down, FP -> R13]	proc0* (var0*)	proc0*	-	-	
↓					
proc1B [lev. up, FP -> R12]	proc1B (var1B)	proc0*	proc1B	-	
↓					
proc2D [lev. up, FP -> R11]	proc2D (var2D)	proc0*	proc1B	proc2D	
↓					
...					

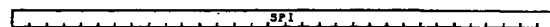
【図350】



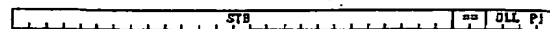
【図352】



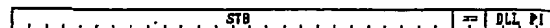
【図351】



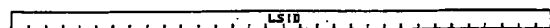
【図353】



【図354】



【図355】



低アドレス↑

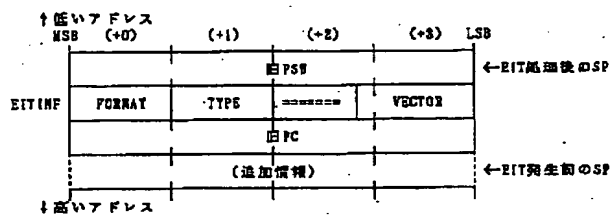
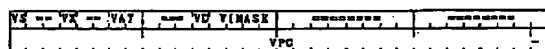
```

CTXB  ─────────→ CSW [32bit]
                SPO [32bit] ─────────→ PSW[32bit]
                SP1 [32bit]  *1, *2    FORMAT/VECTOR[32bit]
                SP2 [32bit]  *1, *2    PC[32bit]
                SP3 [32bit]  *1        ...
                UATB[32bit]  *2        (EIT追加情報) [32×n bit]
                LSID[32bit]  *3        ...
                RO  [32bit]  *4
                ...
                R14 [32bit]  *4
                ...
                (コプロセッサレジスタ退避用)  *5
                ...
                (OS用 - Process ID, Task IDなど)
                ...

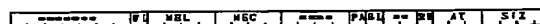
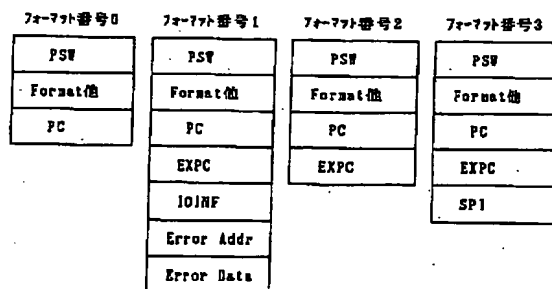
```

高アドレス↓

【 3 5 9】



【图 3 6 2】



【图 3 6 7】

REITでポップされた EIT LIFE中のTYPE	REIT図表に受け付けられる EITのTYPE
1	1-4
2	3-4(2-4ではない)
3	3-4
4	4

IMAGE	記録される DI	許可される外部割り込み
0	—	INT 0 (NMI)
1	DI 0	INT 0 (NMI)
2	DI 0-1	INT 0-1
3	DI 0-2	INT 0-2
4	DI 0-3	INT 0-3
5	DI 0-4	INT 0-4
...		...
13	DI 0-12	INT 0-12
14	DI 0-13	INT 0-13
15	DI 0-14	INT 0-14

【图 3 6 8】

【図363】

No.	Addr	名前	内容	分類	実行	実行
06	+030	予約				
07	+078	予約				
10	+080	DBE	デバッグ例外	完了	2	2
11	+088	BAE	A'27762例外(277A'777)	完了	1	1
			A'27762例外(277A'777以外)	再実行	4	1
12	+090	ATRE	7'62置換例外(277A'777)	完了	1	1
			7'62置換例外(277A'777以外)	再実行	4	1
13	+098	予約	---'5'不在例外			
14	+0A0	PIE	予約命令例外	再実行	4	0
15	+0A8	PIVE	特権命令違反例外	再実行	4	0
18	+0B0	REE	予約値例外	再実行	4	0
17	+0B8	RSFE	予約3777777777例外	再実行	4	0
18	+0C0	予約	---'7'置換例外			
19	+0C8	GAJE	奇数7'62'777'例外	完了	1	2
1A	+0D0	ZDE	7'62置換例外	完了	1	2
18	+0D8	IOE	不正な'777'例外	再実行	4	0
1C	+0E0	予約	---10置換例外			
10	+0E8	LIE	<4>'>命令例外	再実行	4	0
1E	+0F0	予約				
1F	+0F8	TRAP	条件トラップ命令	完了	1	2
20	+100	TRAP	トラップ命令	完了	1	2
21	+108	TRAP	トラップ命令	完了	1	2
22	+178	TRAP	トラップ命令	完了	1	2

【図364】

30	+180	CIE	27' 0777命令例外	再実行	4	0
31	+188	CIE	27' 0777命令例外	再実行	4	0
37	+138	CIE	27' 0777命令例外	再実行	4	0
38	+1C0	予約	---27' 0777実行例外			
39	+1C8	予約	---27' 0777実行例外			
3A	+1D0	予約				
3F	+178	予約				
40	+200	FVEI	固定'外部'外部割り込み	完了	3	0
41	+208	FVEI	固定'外部'外部割り込み	完了	3	0
46	+230	FVEI	固定'外部'外部割り込み	完了	3	0
47	+238	予約	一固定'外部'外部割り込み			
48	+240	予約	一固定'外部'外部割り込み			
4E	+277	予約	一固定'外部'外部割り込み			
4F	+278	予約				
50	+280	D1	遅延割り込み例外	完了	3	0
51	+288	D1	遅延割り込み例外	完了	3	0
5E	+270	D1	遅延割り込み例外	完了	3	0
5F	+278	予約	一遅延2777777777例外			
60	+300	予約				
7F	+378	予約				
80	+400	EI	外部割り込み	完了	3	0
FF	+778	EI	外部割り込み	完了	3	0
100	+800	予約				
1FF	+FF8	予約				

【図365】

【JRNGでring0に入る場合】

命令実行 ⇒ JRNGの処理
 PC, PSWをSPO/SPIに返還
 ↓
 (返還中のエラー発生の場合)
 ↓
 命令再実行を正しく行なうため
 JRNGにより返還された
 SPO/SPI上のPC, PSWを除く
 ↓
 EIT処理 ⇒ ページ不在例外などのEIT起動
 PC, PSWをSPO/SPIに返還
 ↓
 (返還中のエラー発生の場合)
 ↓
 EIT中のEIT ⇒ システムエラー
 動作停止

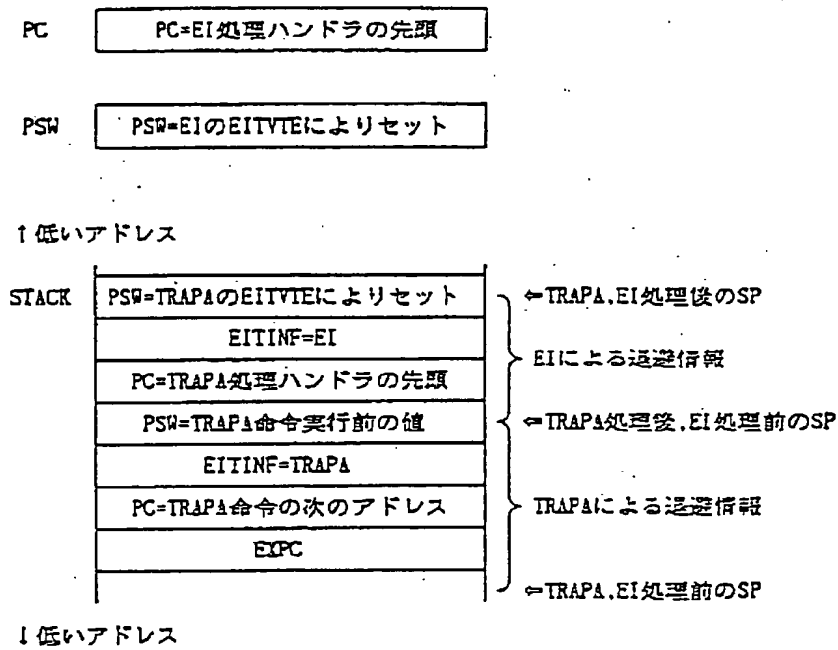
【TRAPAでring0に入る場合】

↓
 ↓
 ↓
 ↓
 ↓
 ↓
 ↓
 ↓
 ↓
 ↓
 TRAPAのEIT起動
 PC, PSWをSPO/SPIに返還
 ↓
 (返還中のエラー発生の場合)
 ↓
 システムエラー
 動作停止

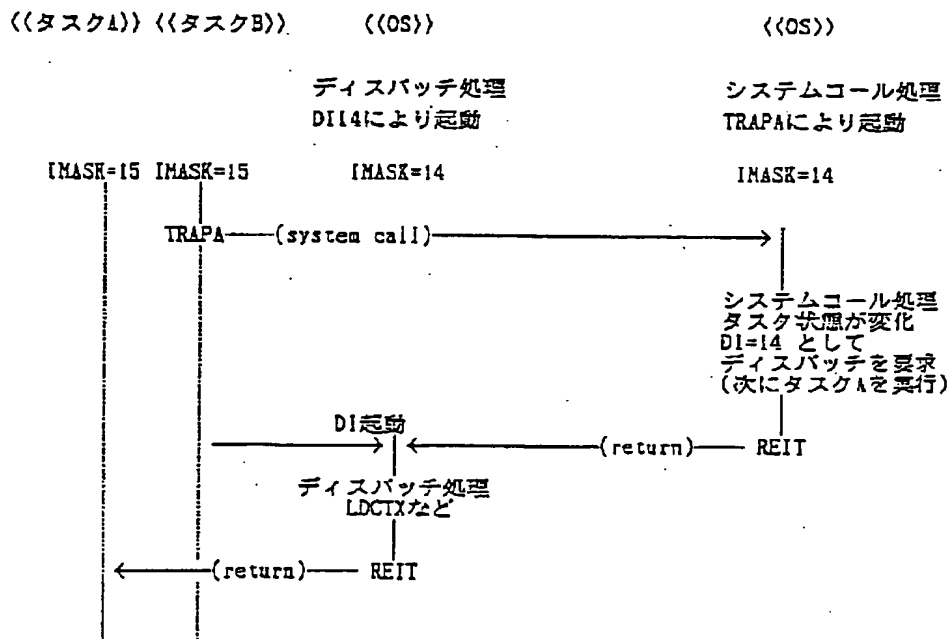
【図400】

命令	src == (PACK, UNPK) ==>	dest	実行後SP
PACKWH @SP+, 0-SP	@(initSP)	@(initSP+4-2)	initSP+2
UNPKWH @SP+, 0-SP	@(initSP)	@(initSP+2-4)	initSP-2

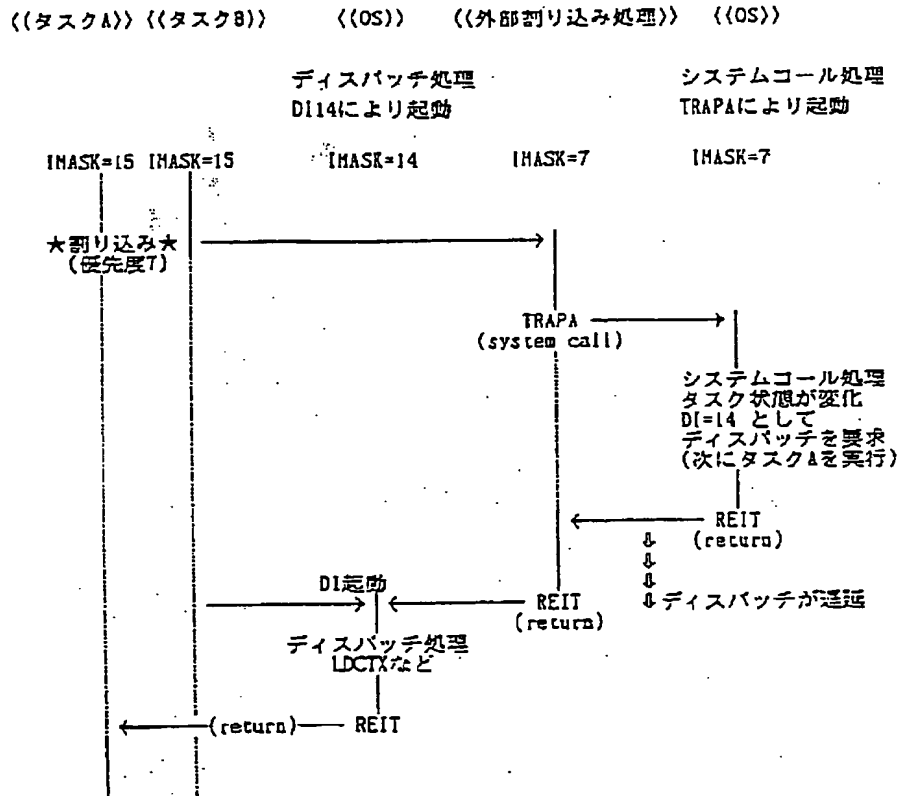
【図366】



【図369】



【図 370】



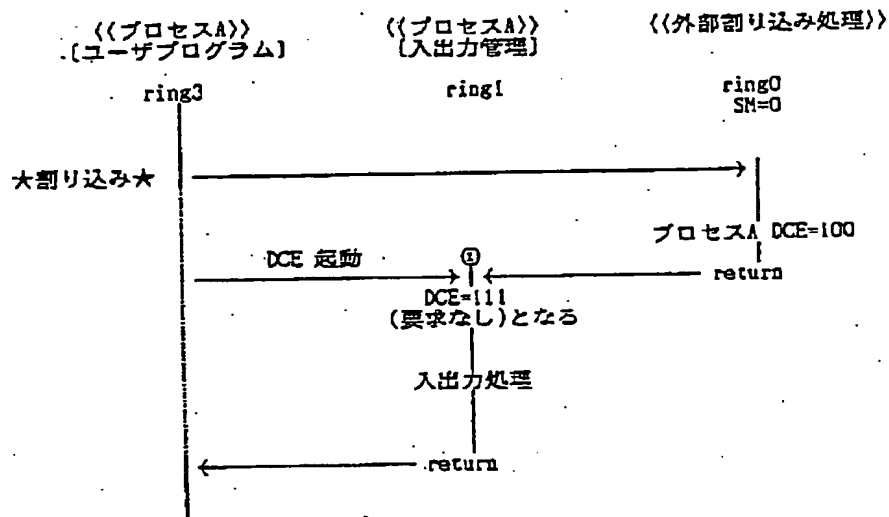
【図 371】

DCE の値	意味
000	無条件 DCE 要求。SH=1 になれば無条件で DCE が起動される。
001	(reserved)
010	(reserved)
011	(reserved)
100	ring1~ring3 になった時に起動される DCE 要求
101	ring2~ring3 になった時に起動される DCE 要求
110	ring3 になった時に起動される DCE 要求
111	DCE 要求なし

【図 3 7 2】

DCE	DI	外部割り込み(EI)
SHRNGの値により pendingとなる	IMASKの値により pendingとなる	IMASKの値により pendingとなる
コンテキスト従属	コンテキスト独立	コンテキスト独立
内部事象と コンテキストとの関係 (ソフトウェア)	内部事象と プロセッサとの関係 (ソフトウェア)	外部事象と プロセッサとの関係 (ハードウェア)

【図 3 7 3】



【図 3 8 1】

{RIE-X}	110100RR ..EaR...	11100*+X ..Ea?bf.	XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX
{RIE-X}	10111111 ..#ib...	11100*+X ..Ea?bf.	XXXXXXXXXXXX	XXXXXXXXXXXX
BFEXT:G	110100RR ..EaR...	111010+X ..EaRbf.	**RRXw**	**RwXd
BFEXT:E	10111111 ..#ib...	111010+X ..EaRbf.	..#n..	**RwXd
BFEXTU:G	110100RR ..EaR...	111011+X ..EaRbf.	**RRXw**	**RwXd
BFEXTU:E	10111111 ..#ib...	111011+X ..EaRbf.	..#n..	**RwXd
ACB:G	110100RR ..EaR...	11110PXX ..EaRX..	**RgHXSS	..#dS8..
ACB:E	10111111 ..#ib...	11110PXX ..EaRX..	**RgHXSS	..#dS8..
SCB:G	110100RR ..EaR...	11111PXX ..EaRX..	**RgNXSS	..#dS8..
SCB:E	10111111 ..#ib...	11111PXX ..EaRX..	**RgNXSS	..#dS8..

【図374】

{00??????}

CMP:L 00RgRwRR 00.ShR..
 MOV:L 00RgRwRR 01.ShR..
 MOV:S 00RgRwRW 10.ShR..
~~MOV:S~~ CMP と MOV が並ぶ場合は、CMP に 0、MOV に 1 が割り当てられる。

AND:R 00RgHw00 1100RgRw
 OR:R 00RgHw01 1100RgRw
 XOR:R 00RgHw10 1100RgRw
 MOVA:R 00RgWp11 1100RgRP#d16.....
 MUL:R 00RgHw00 1101RgRw
 DIV:R 00RgHw01 1101RgRw

⑤ 命令 00????1? 1101????
 ⑥ 命令 00?????? 111?????

{01??????}

CMP:Q 010#3nRR 00.ShR!..
 MOV:Q 011#3nRW 00.ShW..
 ADD:Q 010#3nMM 01.ShM..
 SUB:Q 011#3nMM 01.ShM..
 SHL:Q 010#3nMM 10.ShM..
 SHL:C 011#3cMM 10.ShM..
 SRA:C 011#3cMM 11.ShM..

CMP:I 010000RR 11.ShR!..
 ADD:I 010001MM 11.ShM..
 MOV:I 010010RW 11.ShW..
 SUB:I 010011MM 11.ShM..
 AND:I 010100MM 11.ShM..
 OR:I 010101MM 11.ShM..
 XOR:I 010110MM 11.ShM..
 [R/E] 010111MM 11.ShM..
#IR.....
#IM.....
#IW.....
#IM.....
#IM.....
#IM.....
#IM.....
#IM.....

~~MOV:S~~ CMP と MOV の区別、および ADD と SUB の区別は 2³ ビットで行なう。また、AND、OR、XOR の区別は 2²~2³ ビットで行なう。これは :I フォーマット、:G フォーマットとも共通である。

【図391】

終了条件=検索条件 (=処理続行条件の逆)	オプションのニモニク	N_flag=1 の 条件
< R3.or. ≥ R4	OUTU out of (unsigned)	≥ R4
= R3.or. = R4	OEQ or. equal	= R4
< R3.or. ≥ R4	OUT out of (signed)	≥ R4
= 0	Z zero	= 0 (always)
= R3.or. = 0	ZE zero, equal	= 0

【図417】

ADD, R0, R1

R0 ---- 00110000:00110000 (2)
 +
 R1 ---- 01010101 (2)

 (a) 00110000:10000101 (2)
 R1 ---- (b)(c) 10000101 (2)
 (d)

【図 3 7 5】

[10??????]

Bcc:D	10cccc00 ..#d8...
ADD:L	10RgHw01 00.ShRw.
SUB:L	10RgHw01 01.ShRw.
BSET:Q	100#3z01 10.ShHfg
BCLR:Q	101#3z01 10.ShHfg
BSET1:Q	100#3z01 11.ShHfg
B1ST:Q	101#3z01 11.ShHfg

----- 以下の部分のビット割り当て方針

10????01?	Ea 有り
10xx011?	Ea 有り (LDCTX)
10????111?	imm8 or disp8 有り

10001?1X	ENTER:E,STM (レジスタリスト有り) — ENTER:G も共通
10011?1X	EXIT:E,LDM (レジスタリスト有り) — EXIT:G も共通

JRNG:E と JRNG:G, BSR:8 と JSR のパターンもできる限り共通化した。

JMP	10000010 ..EaA...
ACS	10000011 ..EaA...
POP	10010010 ..EaRL...
PUSHA	1010001S ..EaA...
PUSH	1011001R ..EaRL...
LDCTX	10xx0110 ..EaA!A.
(R1E)	10xx0111 ..EaA!A.
STM	10001010 ..EaGwL.LsRL.....
LDM	1001101R ..EaRcL.L1RL.....
JSR	1010101P ..EaA...
JRNG:G	1011101P ..EaRh!M

【図376】

ENTER:E	1000111X	..#ib...LnXL.....
EXITD:E	1001111X	..#ib...LxXL.....
BRA:D	10101110	..#d8...	
BSR:D	10101111	..#d8...	
JRNG:E	10111110	..#ib...	
①一般形#	10111111	..#ib...	????????? ..Ea?...

[11??????]

CMP:Z	110000SS	..EaR .	
MOV:Z	110001WW	..EaW...	
NEG	110010MM	..EaM...	
NOT	110011MM	..EaM...	
②一般形A	110100RR	..EaR...	????????? ..Ea?...
③一般形B	110101RR	..EaR...	????????? ..Ea?...
④一般形特殊	11011000	..EaA...	????????? ..Ea?...
{RIE}	11011001	xxxxxxxx	
{RIE}	1101101*	xxxxxxxx	
LDPSB	11011100	..EaRh..	

【図382】

[⑤一般形B]

⑤一般形B 110101RR ..EaR... ???????? ..Ea?...

~~~~~ 第二HWの割り当て方針

|          |                                    |
|----------|------------------------------------|
| 00(Rn)?? | 第一HW 'RR'、サイズ指定なし、レジスタ指定あり         |
| 01????SS | 第一HW 'RR'、サイズ指定あり、レジスタ指定なし         |
| 10????0? | 第一HW 'IR'、サイズ指定なし、レジスタ指定なし         |
| 10(Rn)1? | 第一HW 'IR'、サイズ指定なし、レジスタ指定あり         |
| 11(Rn)SS | 第一HW 'IR'、サイズ指定あり、レジスタ指定あり (INDEX) |

|         |          |          |          |                             |
|---------|----------|----------|----------|-----------------------------|
| CSI     | 110101RR | ..EaR... | 00RMC.00 | ..EaMIR.                    |
| {RIE-X} | 110101RR | ..EaR... | 00~~~~01 | ..Ea?...                    |
| CHK     | 110101RR | ..EaR... | 00RgWR1c | ..EaRdR.                    |
| RVBY    | 110101RR | ..EaR... | 010000WW | ..EaW...                    |
| RVBI    | 110101RR | ..EaR... | 010001WW | ..EaW...                    |
| PACZss  | 110101RR | ..EaR... | 010010WW | ..EaW...                    |
| UNPKss  | 110101RR | ..EaR... | 010011WW | ..EaW...      .....#10..... |
| BSCB    | 110101RR | ..EaR... | 0101bdMM | ..EaM...                    |
| DCADJ   | 110101RR | ..EaR... | 011000WW | ..EaW...                    |
| DCADJV  | 110101RR | ..EaR... | 011001WW | ..EaW...                    |
| {RIE-Y} | 110101RR | ..EaR... | 011010WW | ..EaW...                    |
| DCADJY  | 110101RR | ..EaR... | 011011WW | ..EaW...                    |
| {RIE-X} | 110101RR | ..EaR... | 0111~~?? | ..Ea?...                    |

~~~~~ DC7??I 命令のビットパターンを ??1011SS という形に描えた。

【図377】

| | | | |
|----------------|----------|----------|--------------------|
| LDPSM | 11011101 | ..EaRh.. | |
| STPSB | 11011110 | ..EaWh.. | |
| STPSM | 11011111 | ..EaWh.. | |
| coproc1 | 1110**?? | ..Ea?... | ***** |
| coproc2 | 1111**** | ***** | |
| 【①② 一般形#/一般形A】 | | | |
| ②一般形A | 110100RR | ..EaR... | ????????? ..Ea?... |
| ③一般形# | 10111111 | ..#ib... | ????????? ..Ea?... |
| ADD:G | 110100RR | ..EaR... | 000000MM ..EaM... |
| ADD:E | 10111111 | ..#ib... | 000000MM ..EaM... |
| ADDU:G | 110100RR | ..EaR... | 000001MM ..EaM... |
| ADDU:E | 10111111 | ..#ib... | 000001MM ..EaM... |
| SUB:G | 110100RR | ..EaR... | 000010MM ..EaM... |
| SUB:E | 10111111 | ..#ib... | 000010MM ..EaM... |
| SUBU:G | 110100RR | ..EaR... | 000011MM ..EaM... |
| SUBU:E | 10111111 | ..#ib... | 000011MM ..EaM... |

**** signed の命令と unsigned 命令の区別は、2² のビットによって行なわれる。これは、ADD, SUB, MUL, DIV, REM, CMP, MOV, BFCMP, BFINS に共通である。

| | | | |
|---------|----------|----------|-------------------|
| ADDX:G | 110100RR | ..EaR... | 000100MM ..EaM... |
| ADDX:E | 10111111 | ..#ib... | 000100MM ..EaM... |
| ADDX:G | 110100RR | ..EaR... | 000101MM ..EaM... |
| ADDX:E | 10111111 | ..#ib... | 000101MM ..EaM... |
| SUBX:G | 110100RR | ..EaR... | 000110MM ..EaM... |
| SUBX:E | 10111111 | ..#ib... | 000110MM ..EaM... |
| SUBDX:G | 110100RR | ..EaR... | 000111MM ..EaM... |
| SUBDX:E | 10111111 | ..#ib... | 000111MM ..EaM... |

【図385】

| | | | | |
|---------|----------|----------|-----------|------------------------|
| PSTLB | 000000+ | 1111P111 | 0-pttc- | ..EaA... |
| (R/E-X) | 000000+X | 1111P111 | 1-***0*?? | ..Ea?... |
| SHXL | 000000+X | 1111-111 | 1-010+ | ..EaMX.. |
| SHXR | 000000+X | 1111-111 | 1-110+ | ..EaMX.. |
| ENTER:G | 000000+X | 1111P111 | 1-011SS | ..EaR!M.LxXL..... |
| EXITD:G | 000000+X | 1111P111 | 1-111SS | ..EaR!M.LxXL..... |

**** EaR!M では Rn と #imm_data のみが許される。EaR!M のサイズは SS で指定される。
LxRL, LxRL で返送。復帰するレジスタのサイズは X で指定される。

| | | | |
|-------|----------|----------|---------------|
| BVPAT | 000001+X | 1111P111 | |
| (R/E) | 00001**X | 1111P111 | |
| BVSCR | 0001bd+X | 1111P111 | |
| BRA:G | 001000SS | 1111P111 |#dS..... |
| SSTR | 001001SS | 1111P111 | |
| BSR:G | 00101QSS | 1111P111 |#dS..... |
| BVHAP | 0011bQOX | 1111P111 | |
| BVCPY | 0011bQIX | 1111P111 | |

【図378】

| | | |
|---------|-------------------|-------------------|
| AND:G | 110100RR ..EaR... | 001000HM ..EaH... |
| AND:E | 10111111 ..#ib... | 001000HM ..EaH... |
| OR:G | 110100RR ..EaR... | 001001HM ..EaH... |
| OR:E | 10111111 ..#ib... | 001001HM ..EaH... |
| XOR:G | 110100RR ..EaR... | 001010HM ..EaH... |
| XOR:E | 10111111 ..#ib... | 001010HM ..EaH... |
| DCX:G | 110100RR ..EaR... | 001011HM ..EaH... |
| DCX:E | 10111111 ..#ib... | 001011HM ..EaH... |
| | | |
| SHL:G | 110100RR ..EaR... | 001100HM ..EaH... |
| SRL:E | 10111111 ..#ib... | 001100HM ..EaH... |
| SHA:G | 110100RR ..EaR... | 001101HM ..EaH... |
| SRA:E | 10111111 ..#ib... | 001101HM ..EaH... |
| ROT:G | 110100RR ..EaR... | 001110HM ..EaH... |
| ROT:E | 10111111 ..#ib... | 001110HM ..EaH... |
| {RIE-X} | 110100RR ..EaR... | 001111HM ..EaH... |
| {RIE-X} | 10111111 ..#ib... | 001111HM ..EaH... |
| | | |
| MUL:G | 110100RR ..EaR... | 010000HM ..EaH... |
| MUL:E | 10111111 ..#ib... | 010000HM ..EaH... |
| MULU:G | 110100RR ..EaR... | 010001HM ..EaH... |
| MULU:E | 10111111 ..#ib... | 010001HM ..EaH... |
| DIV:G | 110100RR ..EaR... | 010010HM ..EaH... |
| DIV:E | 10111111 ..#ib... | 010010HM ..EaH... |
| DIVU:G | 110100RR ..EaR... | 010011HM ..EaH... |
| DIVU:E | 10111111 ..#ib... | 010011HM ..EaH... |
| | | |
| {RIE-X} | 110100RR ..EaR... | 01010*HM ..EaH... |
| {RIE-X} | 10111111 ..#ib... | 01010*HM ..EaH... |
| REM:G | 110100RR ..EaR... | 010110HM ..EaH... |
| REM:E | 10111111 ..#ib... | 010110HM ..EaH... |
| REMU:G | 110100RR ..EaR... | 010111HM ..EaH... |
| REMU:E | 10111111 ..#ib... | 010111HM ..EaH... |

* REM, REMU と DIV, DIVU のパターンをできるだけ共通化した。

【図397】

| 命令 | 動作 | 実行後SP |
|-----------|--|----------|
| PUSHA @SP | srcaddr: initSP
initSP ==> @(initSP-4)
[MOVA @SP, @-SP と同じ] | initSP-4 |
| JSR @SP | newpc: initSP
initPC ==> @(initSP-4)
initSP ==> PC
[initSP-4 ==> PC ではない] | initSP-4 |

【図379】

| | | |
|----------|-------------------|-------------------|
| DCADD:G | 110100RR ..EaR... | 011000MM ..EaM... |
| DCADD:E | 10111111 ..#ib... | 011000MM ..EaM... |
| DCADDU:G | 110100RR ..EaR... | 011001MM ..EaM... |
| DCADDU:E | 10111111 ..#ib... | 011001MM ..EaM... |
| DCSUB:G | 110100RR ..EaR... | 011010MM ..EaM... |
| DCSUB:E | 10111111 ..#ib... | 011010MM ..EaM... |
| DCSUBU:G | 110100RR ..EaR... | 011011MM ..EaM... |
| DCSUBU:E | 10111111 ..#ib... | 011011MM ..EaM... |

| | | |
|---------|-------------------|-------------------|
| (RIE-X) | 110100RR ..EaR... | 0111**MM ..EaM... |
| (RIE-X) | 10111111 ..#ib... | 0111**MM ..EaM... |

| | | |
|--------|-------------------|-------------------|
| CMP:G | 110100RR ..EaR... | 100000SS ..EaR!! |
| CMP:E | 10111111 ..#ib... | 100000SS ..EaR!! |
| CMPI:G | 110100RR ..EaR... | 100001SS ..EaR!! |
| CMPI:E | 10111111 ..#ib... | 100001SS ..EaR!! |
| MOV:G | 110100RR ..EaR... | 100010WW ..EaW... |
| MOV:E | 10111111 ..#ib... | 100010WW ..EaW... |
| MOVU:G | 110100RR ..EaR... | 100011WW ..EaW... |
| MOVU:E | 10111111 ..#ib... | 100011WW ..EaW... |

XXXXXXXXX CMP, CMPI, BFCMP, BFCMPI, DCCMP, DCCMPI および MOV, MOVU, LDP, LDC, BFINS, BFINSU のパターンをできるだけ揃えた。

| | | |
|----------|-------------------|------------------|
| DCCMP:G | 110100RR ..EaR... | 100100SS ..EaR!! |
| DCCMP:E | 10111111 ..#ib... | 100100SS ..EaR!! |
| DCCMPI:G | 110100RR ..EaR... | 100101SS ..EaR!! |
| DCCMPI:E | 10111111 ..#ib... | 100101SS ..EaR!! |
| LDC:G | 110100RR ..EaR... | 100110WW ..EaW? |
| LDC:E | 10111111 ..#ib... | 100110WW ..EaW? |
| LDP:G | 110100RR ..EaR... | 100111WW ..EaW? |
| LDP:E | 10111111 ..#ib... | 100111WW ..EaW? |

XXXXXXXXX 特殊空間の区別(LDPとLDC)は2*2ビットで行なう。STPとSTCの区別も同じである。

【図396】

| 命令 | 動作 | 実行後SP |
|---------------|---|----------|
| PUSH SP | src: initSP
initSP == @ (initSP-4)
(フラグ変化を除けば MOV SP, @-SP と同じ) | initSP-4 |
| PUSH @SP+ | 《モデル上は以下の動作をするが、実際は RIE》
src: @initSP
@initSP == @ (initSP+4-4)
(フラグ変化を除けば MOV @SP+, @-SP と同じ) | initSP |
| PUSH @(d, SP) | src: @(d+initSP)
@(d+initSP) == @ (initSP-4)
(フラグ変化を除けば MOV @(d, SP), @-SP と同じ) | initSP-4 |

【図380】

| | | |
|------------|-------------------|--|
| BSETI:G | 110100RR ..EaR... | 101000BB ..EaMfi. |
| BSETI:E | 10111111 ..#ib... | 101000BB ..EaMfi. |
| BCLRI:G | 110100RR ..EaR... | 101001BB ..EaMfi. |
| BCLRI:E | 10111111 ..#ib... | 101001BB ..EaMfi. |
| (RIE-X) | 110100RR ..EaR... | 101010?? ..Ea?... |
| (RIE-X) | 10111111 ..#ib... | 101010?? ..Ea?... |
| DOCHPX:G | 110100RR ..EaR... | 101011SS ..EaRfI. |
| DOCHPX:E | 10111111 ..#ib... | 101011SS ..EaRfI. |
| | | |
| BSET:G | 110100RR ..EaR... | 101100BB ..EaMf.. |
| BSET:E | 10111111 ..#ib... | 101100BB ..EaMf.. |
| BCLR:G | 110100RR ..EaR... | 101101BB ..EaMf.. |
| BCLR:E | 10111111 ..#ib... | 101101BB ..EaMf.. |
| BNOT:G | 110100RR ..EaR... | 101110BB ..EaMf.. |
| BNOT:E | 10111111 ..#ib... | 101110BB ..EaMf.. |
| BTST:G | 110100RR ..EaR... | 101111BB ..EaRf.. |
| BTST:E | 10111111 ..#ib... | 101111BB ..EaRf.. |
| | | |
| BFCMP:G:R | 110100RR ..EaR... | 110000+X ..EaRbf. RRX RRXs |
| BFCMP:E:R | 10111111 ..#ib... | 110000+X ..EaRbf. #6n RRXs |
| BFCMPU:G:R | 110100RR ..EaR... | 110001+X ..EaRbf. RRX RRXs |
| BFCMPU:E:R | 10111111 ..#ib... | 110001+X ..EaRbf. #6n RRXs |
| BFINS:G:R | 110100RR ..EaR... | 110010+X ..EaMbf. RRX RRXs |
| BFINS:E:R | 10111111 ..#ib... | 110010+X ..EaMbf. #6n RRXs |
| BFINSU:G:R | 110100RR ..EaR... | 110011+X ..EaMbf. RRX RRXs |
| BFINSU:E:R | 10111111 ..#ib... | 110011+X ..EaMbf. #6n RRXs |
| | | |
| BFCMP:G:I | 110100RR ..EaR... | 110100+X ..EaRbf. RRXwSS #iSB |
| BFCMP:E:I | 10111111 ..#ib... | 110100+X ..EaRbf. #6n SS #iSB |
| BFCMPU:G:I | 110100RR ..EaR... | 110101+X ..EaRbf. RRXwSS #iSB |
| BFCMPU:E:I | 10111111 ..#ib... | 110101+X ..EaRbf. #6n SS #iSB |
| BFINS:G:I | 110100RR ..EaR... | 110110+X ..EaMbf. RRXwSS #iSB |
| BFINS:E:I | 10111111 ..#ib... | 110110+X ..EaMbf. #6n SS #iSB |
| BFINSU:G:I | 110100RR ..EaR... | 110111+X ..EaMbf. RRXwSS #iSB |
| BFINSU:E:I | 10111111 ..#ib... | 110111+X ..EaMbf. #6n SS #iSB |

【図399】

| 命令 | src | dest | 実行後SP |
|---------------------|-------------|-------------------|----------|
| MOV.W @SP+, @-SP | @initSP | ==> @initSP | initSP |
| MOV.W @SP+, @(d,SP) | @initSP | ==> @(d+initSP+4) | initSP+4 |
| MOV.W @(d,SP), @-SP | @(d+initSP) | ==> @(initSP-4) | initSP-4 |
| MOV.W SP, @-SP | initSP | ==> @(initSP-4) | initSP-4 |
| MOV.W @SP+, SP | @initSP | ==> SP | @initSP |

【図383】

| | | |
|----------|---------------------|----------------------|
| LDAT | 110101!R ..EaR... | 10pttt00 ..EaA... |
| (RIE-X) | 110101!R ..EaR... | 10xxxx01 ..Ea?... |
| MVLY | 110101!R ..EaR... | 10RGR10 ..EaNR... |
| DVY | 110101!R ..EaR... | 10RGR11 ..EaNR... |
| INDEX | 110101!R ..EaR... | 11RGRSS ..EaR2... |
| 〔㊟一般形特殊〕 | | |
| ㊟一般形特殊 | 11011000 ..EaA... | ???????? ..Ea?... |
| (RIE-X) | 11011000 ..EaA... | 0xxxxx?? ..Ea?... |
| STATE | 11011000 ..EaA... | 100ctt+W ..EaW!S... |
| (RIE-X) | 11011000 ..EaA... | 101000?? ..Ea?... |
| NOYPA | 11011000 ..EaA... | 101001+W ..EaW!S... |
| STC | 11011000 ..EaR... | 101010WW ..EaW... |
| STP | 11011000 ..EaR... | 101011WW ..EaW... |
| QDEL | 11011000 ..EaRqP... | 101100+W ..EaW!S... |
| NOVA:G | 11011000 ..EaA... | 101101+W ..EaW... |
| QINS | 11011000 ..EaMaP... | 101110+- ..EaMaP2... |
| (RIE-X) | 11011000 ..Ea?... | 101111?? ..Ea?... |
| (RIE-X) | 11011000 ..EaA... | 11xxxx?? ..Ea?... |
| 〔㊟ 雑命令〕 | | |
| ㊟ 雑命令 | 00????1? 1101???? | |
| (RIE) | 00xxxx10 1101xxxx | |
| ACB:R | 00RGR11 1101P000 | --RGRSS ..#dSB... |
| ACB:Q | 00RGR11 1101P001 | ..#R...SS ..#dSB... |
| SCB:R | 00RGR11 1101P010 | --RGRSS ..#dSB... |
| SCB:Q | 00RGR11 1101P011 | ..#R...SS ..#dSB... |
| TRAP | 00cccc11 1101P100 | |
| TRAPA | 00#4z..11 1101P101 | |

【図403】

| 命令 | src1 | src2 | 実行後SP |
|--------------------|-----------|---------------|----------|
| CHP.W @SP+, @SP+ | @(initSP) | @(initSP+4) | initSP+8 |
| CHP.W @SP+, @d,SP) | @(initSP) | @(d+initSP+4) | initSP+4 |
| CHP.W @SP+, SP | @(initSP) | initSP+4 | initSP+4 |
| CHP.W SP, @SP+ | initSP | @(initSP) | initSP+4 |

【図384】

==== 以下の部分の割り当て方針

```

00??0011 1101P110    LVreserved
00??1011 1101P110    一般命令
00??0111 1101P110    特権命令 (STCTX)
00??1111 1101P110    特権命令

```

```

LVreserved    00==0011 1101*110
STCTX         00xx0111 1101P110

```

```

PIB           00001011 1101P110
NOP           00011011 1101-110
RTS           00101011 1101P110
RRNG          00111011 1101P110

```

```

WAIT          00001111 1101-110    .....#ih.....
REIT          00101111 1101P110
PTLB          00p11111 1101P110

```

```

(RIE)         00====11 1101P111

```

【⑥ 雑命令】

```

⑥雑命令      007777?? 111777??

```

```

SCHP          00eeeeSS 1110P0Qb
SMOV          00eeeeSS 1110P1Qb
QSCH          00eeeeSS 1111P0ab
SSCH          00eeeeSS 1111P10c

```

```

Bcc:G         00ccccSS 1111P110    .....#dS.....

```

==== 以下の部分の割り当て方針

```

000777?? 1111P111    第一HW の 2'9 のビットは常に '+'
001777?? 1111P111    第一HW の 2'9 のビットは 0/1 の両方可

```

【図401】

| 命令 | 動作 | 実行後SP |
|------------------|--|----------|
| MOV.W SP, @-SP | initSP ==> @(initSP-4) | initSP-4 |
| STC.W @sp0, @-SP | <<モデル上は以下の動作をするが、実際はインプリメント依存>>
initSP-4 ==> @(initSP-4)
(sp0 は SPO を制御レジスタとして指定したもの)
(initSP ==> @(initSP-4) ではない) | initSP-4 |
| MOVA.W @SP, @-SP | initSP ==> @(initSP-4)
(initSP-4 ==> @(initSP-4) ではない) | initSP-4 |

【図386】

```

#3c  SHA:C, SHL:C
#3d  ADD:Q, CMP:Q, MOV:Q, SHL:Q, SUB:Q
#3z  BCLR:Q, BSET:Q, BSETI:Q, BTST:Q
#4z  TRAPA
#6a  ACB:Q, BFCMP:E:I, BFCMP:E:R, BFCMPU:E:I, BFCMPU:E:R, BFEXT:E, BFEXTU:E, BFINS:E:I,
      BFINS:E:R, BFINSU:E:I, BFINSU:E:R
#6z  SCB:Q
#d1b NOVA:B
#d8  BRA:D, BSR:D, Bcc:D
#dS  BRA:G, BSR:G, Bcc:G
#dS8 ACB:E, ACB:G, ACB:Q, ACB:R, SCB:E, SCB:G, SCB:Q, SCB:R
#iX  ADD:I, AND:I, OR:I, SUB:I, XOR:I, (RIE)
#iR  CMP:I
#iS8 BFCMP:E:I, BFCMP:G:I, BFCMPU:E:I, BFCMPU:G:I, BFINS:E:I, BFINS:G:I, BFINSU:E:I,
      BFINSU:G:I
#iQ  MOV:I, UNPKss
#ib  ACB:E, ADD:E, ADDX:E, ADDU:E, ADDX:G, AND:E, BCLR:E, BCLR:I, BFCMP:E:I, BFCMP:E:R,
      BFCMPU:E:I, BFCMPU:E:R, BFEXT:E, BFEXTU:E, BFINS:E:I, BFINS:E:R, BFINSU:E:I, BFINSU:E:R,
      BNOT:E, BSET:E, BSETI:E, BTST:E, CMP:E, CMPU:E, DCADD:E, DCADDU:E, DCCMP:E, DCCMPU:
      E, DCCMPX:E, DCSUB:E, DCSUBU:E, DCX:E, DIV:E, DIVU:E, ENTER:E, EXITD:E, JRCG:E, LDC:E,
      LDP:E, MOV:E, MOVU:E, MUL:E, MULU:E, OR:E, REM:E, REMU:E, ROT:E, SCB:E, SHA:E, SHL:E,
      SUB:E, SUBDX:E, SUBU:E, SUBX:E, XOR:E
#ih  WAIT
EaA  ACS, JMP, JSR, LDATZ, NOVA:C, MOVPA, PSTLB, PUSH, STATE
EaA1A LDCTX, (RIE)
EaH  ADD:E, ADD:G, ADDX:E, ADDX:G, ADDU:E, ADDU:G, ADDX:E, ADDX:G, AND:E, AND:G, BSCE,
      DCADD:E, DCADD:G, DCADDU:E, DCADDU:G, DCSUB:E, DCSUB:G, DCSUBU:E, DCSUBU:G, DCX:E,
      DCX:G, DIV:E, DIV:G, DIVU:E, DIVU:G, MUL:E, MUL:G, MULU:E, MULU:G, NEG, NOT, OR:E, C,
      G, REM:E, REM:G, REMU:E, REMU:G, ROT:E, ROT:G, SHA:E, SHA:G, SHL:E, SHL:G, SUB:E, SUB:
      G, SUBDX:E, SUBDX:G, SUBU:E, SUBU:G, SUBX:E, SUBX:G, XOR:E, XOR:G

```

【図389】

| ニモニック | 意味 | 条件 | cccc |
|-------|---------------------------------|---------------------|------|
| XS | X_flag set | X | 0000 |
| XC | X_flag clear | \bar{X} | 0001 |
| EQ | equal / Z_flag set | Z | 0010 |
| NE | not equal / Z_flag clear | \bar{Z} | 0011 |
| LT | less than / L_flag set | L | 0100 |
| GE | greater or equal / L_flag clear | \bar{L} | 0101 |
| LE | less or equal | L+Z | 0110 |
| GT | greater than | $\bar{L} * \bar{Z}$ | 0111 |
| VS | V_flag set | V | 1000 |
| VC | V_flag clear | \bar{V} | 1001 |
| MS | minus / M_flag set | M | 1010 |
| MC | plus / M_flag clear | \bar{M} | 1011 |
| FS | F_flag set | F | 1100 |
| FC | F_flag clear | \bar{F} | 1101 |
| (RIE) | | | 1110 |
| (RIE) | | | 1111 |

【図 3 8 7】

EaMR DIVX, MULX
 EaMY SEIL, SEXR
 EaMbf BFINS:E:I, BFINS:E:R, BFINS:G:I, BFINS:G:R, BFINSU:E:I, BFINSU:E:R, BFINSU:G:I, BFINSU:G:R
 EaMf BCLR:E, BCLR:G, BNOT:E, BNOT:G, BSET:E, BSET:G
 EaMfi BCLRI:E, BCLRI:G, BSETI:E, BSETI:G
 EaMiR CSI
 EaMqP QINS
 EaMqP2 QINS
 EaR ACB:G, ADDX:G, ADDX:G, ADDU:G, ADDI:G, AND:G, BCLR:G, BCLRI:G, BFCMP:G:I, BFCMP:G:R, BFCMPU:G:I, BFCMPU:G:R, BFEXT:G, BFEXTU:G, BFINS:G:I, BFINS:G:R, BFINSU:G:I, BFINSU:G:R, BNOT:G, BSCH, BSET:G, BSETI:G, BIST:G, CHK, CMP:G, CMPU:G, CSI, DCADD:G, DCADOU:G, DCADJ, DCADJU, DCADJX, DCCMP:G, DCCMPU:G, DCCMPX:G, DCSUB:G, DCSUBU:G, DCX:G, DIV:G, DIVU:G, DIVX, INDEX, LDATE, LDC:G, LDP:G, MOV:G, MOVU:G, MUL:G, Mulu:G, MULX, OR:G, PACKss, REM:G, REMU:G, ROT:G, RVBI, RVBY, SCB:G, SHA:G, SHL:G, SUB:G, SUBDX:G, SUBG:G, SUBX:G, UNPKss, XOR:G
 EaRII CMP:E, CMP:G, CMP:Z, CMPU:E, CMPU:G, DCCMP:E, DCCMP:G, DCCMPU:E, DCCMPU:G, DCCMPX:E, DCCMPX:G
 EaRIM ENTER:G, EXITD:G
 EaRi STC, STP
 EaR2 INDEX
 EaRL PUSH
 EaRX ACB:E, ACB:G, SCB:E, SCB:G
 EaRbf BFCMP:E:I, BFCMP:E:R, BFCMP:G:I, BFCMP:G:R, BFCMPU:E:I, BFCMPU:E:R, BFCMPU:G:I, BFCMPU:G:R, BFEXT:E, BFEXT:G, BFEXTU:E, BFEXTU:G
 EaRdR CHK
 EaRf BIST:E, BIST:G
 EaRh LDPSB, LDPSM
 EaRbIM JRNG:G
 EaRmL LDM
 EaRqP QDEL
 EaW DCADJ, DCADJU, DCADJX, MOV:E, MOV:G, MOV:Z, MOV:G, MOVU:E, MOVU:G, PACKss, RVBI, RVBY, STC, STP, UNPKss

【図 4 0 4】

| 命令 | 動作 |
|----------------------|--|
| INDEX @SP+, @SP+, SP | |
| | indexsize: @initSP |
| | subscript: @(initSP+4) |
| | EX.の直前でのxreg: initSP+8 |
| | $(initSP+8) \times @initSP + @(initSP+4) \Rightarrow SP$ |

【図388】

```

EaWIS  MOVPA, QDEL, STATE
EaW%   LDC:E, LDC:G, LDP:E, LDP:G
EaWL   POP
EaWh   STPSB, STPSH
EaWAL  STM
LIRL   LDM
LaXL   ENTER:E, ENTER:G
LsWL   STM
LxXL   EXITD:E, EXITD:G
RMC     CSI
RRXs    BFCNP:E:R, BFCNP:G:R, BFCNPU:E:R, BFCNPU:G:R, BFINS:E:R, BFINS:G:R, BFINSU:E:R,
        BFINSU:G:R
RRXw    BFCNP:G:I, BFCNP:G:R, BFCNPU:G:I, BFCNPU:G:R, BFEXT:G, BFEXTU:G, BFINS:G:I, BFINS:G:R,
        BFINSU:G:I, BFINSU:G:R
RWXd    BFEXT:E, BFEXT:G, BFEXTU:E, BFEXTU:G
RgHR    DIYI, INDEX
RgHX    ACB:E, ACB:G, SCB:E, SCB:G
RgHw    ACB:Q, ACB:R, ADD:L, AND:R, DIV:R, MUL:R, OR:R, SCB:Q, SCB:R, SUB:L, XOR:R
RgRP    MOVA:R
RgRw    ACB:R, AND:R, CNP:L, DIV:R, MOV:S, MUL:R, OR:R, SCB:R, XOR:R
RgWP    MOVA:R
RgWR    CRR, MULX
RgWw    MOV:L
ShM     ADD:I, ADD:Q, AND:I, OR:I, SHA:C, SEL:C, SHL:Q, SUB:I, SUB:Q, XOR:I, (RIE)
ShHfq   BCLR:Q, BSET:Q
ShHfqI  BSETI:Q
ShR     CNP:L, MOV:L
ShR!!   CNP:I, CNP:Q
ShRfq   BTST:Q
ShRw    ADD:L, SUB:L
ShW     MOV:I, MOV:Q, MOV:S

```

【図406】

| 命令 | src | destから読む値 | 実行後SP |
|-------|---|---------------|------------|
| ADD.W | @SP+,@(d.SP) @initSP | @(d+initSP+4) | initSP+4 |
| | (@(d+initSP+4)+@initSP ==) @(d+initSP+4) を実行) | | |
| ADD.W | @SP+,SP @initSP | initSP+4 | (initSP+4) |
| | ((initSP+4)+@initSP ==) SP を実行) | | |
| | | | +@initSP |

【図390】

| 終了条件=検索条件
(=処理続行条件の逆) | オプションのニモニック | | | eeee |
|--------------------------|-------------|-----------------------------|----------|------|
| < R3 | LTU | less than (unsigned) | | 0000 |
| ≥ R3 | GEU | greater or equal (unsigned) | | 0001 |
| = R3 | EQ | equal | | 0010 |
| ≠ R3 | NE | not equal | | 0011 |
| < R3 | LT | less than (signed) | | 0100 |
| ≥ R3 | GE | greater or equal (signed) | | 0101 |
| 終了条件なし | N | never (またはオプションなし) | | 0110 |
| | (RIE) | | | 0111 |
| < R3.or. ≥ R4 | OUTU | out of (unsigned) | <<(L2)>> | 1000 |
| ≥ R3.and. < R4 | WINU | within (unsigned) | <<(L2)>> | 1001 |
| = R3.or. = R4 | OEQ | or, equal | <<(L2)>> | 1010 |
| ≠ R3.and. ≠ R4 | ANE | and, not equal | <<(L2)>> | 1011 |
| < R3.or. ≥ R4 | OUT | out of (signed) | <<(L2)>> | 1100 |
| ≥ R3.and. < R4 | WIN | within (signed) | <<(L2)>> | 1101 |
| = 0 | Z | zero | <<(L2)>> | 1110 |
| = R3.or. = 0 | ZE | zero, equal | <<(L2)>> | 1111 |

【図392】

src=0, dest=0 の演算結果を bit0 に。
 src=0, dest=1 の演算結果を bit1 に。
 src=1, dest=0 の演算結果を bit2 に。
 src=1, dest=1 の演算結果を bit3 に入れる。

| | | | |
|------|-----|-----------|---------------------------|
| 0000 | F | False | 0 ==> dest |
| 0001 | NAN | NotAndNot | ~dest .and. ~src ==> dest |
| 0010 | AN | AndNot | dest .and. ~src ==> dest |
| 0011 | NS | NotSrc | ~src ==> dest |
| 0100 | NA | NotAnd | ~dest .and. src ==> dest |
| 0101 | ND | NotDest | ~dest ==> dest |
| 0110 | X | Xor | dest .xor. src ==> dest |
| 0111 | NON | NotOrNot | ~dest .or. ~src ==> dest |
| 1000 | A | And | dest .and. src ==> dest |
| 1001 | NX | NotXor | ~dest .xor. src ==> dest |
| 1010 | D | Dest | dest ==> dest |
| 1011 | ON | OrNot | dest .or. ~src ==> dest |
| 1100 | S | Src | src ==> dest |
| 1101 | NO | NotOr | ~dest .or. src ==> dest |
| 1110 | O | Or | dest .or. src ==> dest |
| 1111 | T | True | 1 ==> dest |

【図393】

| | 一般① | Rn | #imm | ESP+ | E-SP | 付加 | (対象命令) |
|--------|-----|--------|------|------|------|----|---|
| EaA | ○ | × | × | × | × | ○ | ACS, JMP, JSR, LDATE,
MOVA:G, PUSH, A,
MOVPA, PSTLB, STATE,
LDCTX |
| EaA A | ○ | × | × | × | × | × | |
| EaM | ○ | ○ | × | × | × | ○ | ADD:E, ADD:G, DIV:E,
DIV:G, DIVU:E, DIVU:G,
SHA:E, SHA:G.....多数 |
| ShM | | | | | | | ADD:I, ADD:Q, SHA:C,
OR:I, AND:I, SHL:Q,
SHL:C, SUB:I, SUB:Q,
XOR:I |
| EaMX | | | | | | | SHXL, SEXR |
| EaMR | | | | | | | DIVY, MULX |
| EaMf | ○ | ○② | × | × | × | ○ | BCLR:E, BCLR:G,
BNOT:E, BNOT:G,
BSET:E, BSET:G |
| ShMfq | ○ | ○ | × | × | × | ○ | BCLR:Q, BSET:Q |
| EaMbf | ○ | <<L2>> | × | × | × | ○ | BFINS:E:I, BFINS:E:R,
BFINSU:E:I, BFINSU:E:R,
BFINS:G:I, BFINS:G:R,
BFINSU:G:I, BFINSU:G:R |
| EaMfi | ○ | × | × | × | × | ○ | BCLRI:E, BCLRI:G,
BSETI:E, BSETI:G |
| ShMfqi | ○ | × | × | × | × | ○ | BSETI:Q |
| EaMfiR | ○ | × | × | × | × | ○ | CSI |
| EaMqP | ○ | × | × | × | × | ○ | QINS |
| EaMqP2 | | | | | | | QINS |
| | 一般 | Rn | #imm | ESP+ | E-SP | 付加 | (対象命令) |

【図407】

| 命令 | 動作 |
|------------------------|--------------------------|
| [CS[comp,update,dest] | |
| CSI SP,@SP+,@(d,SP) | |
| | EX.の直前でのcomp: initSP+4 |
| | update: @initSP |
| | dest: @(d+initSP+4) |
| | (dest: @(d+initSP) ではない] |

【図394】

| | 一般 | Rn | #imm | ESP+ | ESP- | 付加 | (対象命令) |
|--------|----|----------|------|------|------|----|--|
| EaR | ○ | ○ | ○ | ○ | × | ○ | ACB:G, ADD:G, ADDDX:G,
ADDU:G, ADDX:G, AND:G,
BCLR:G, BSET:G,...多数 |
| EaRh | | | | | | | LDPSB, LDPSH |
| SHR | | | | | | | CMP:L, MOV:L |
| ShRw | | | | | | | ADD:L, SUB:L |
| EaR2 | ○ | ○ | ○③ | ○ | × | ○ | INDEX |
| EaRX | | | | | | | ACB:E, ACB:G,
SCB:E, SCB:G |
| EaRmL | ○ | × | × | ○ | × | × | LDM |
| EaRL | ○ | ○ | ○ | × | × | ○ | PUSH |
| EaR!I | ○ | ○ | × | ○ | × | ○ | CMP:E, CMP:G,
CMPU:E, CMPU:G, CMP:Z |
| ShR!I | | | | | | | CMP:I, CMP:Q |
| EaR% | ○ | × | × | × | × | ○ | STC, STP |
| EaRdR | ○ | × | × | × | × | ○ | CHR |
| EaRqP | ○ | × | × | × | × | ○ | QDEL |
| EaRf | ○ | ○③ | × | × | × | ○ | BTST:E, BTST:G |
| ShRfq | ○ | ○ | × | × | × | ○ | BTST:Q |
| EaRbf | ○ | <<(L2)>> | × | × | × | ○ | BFCMP:E:I, BFCMPU:E:I,
BFCMP:E:R, BFCMPU:E:R,
BFEXT:E, BFEXTU:E
BFCMP:G:I, BFCMPU:G:I,
BFCMP:G:R, BFCMPU:G:R,
BFEXT:G, BFEXTU:G |
| EaR!M | × | ○ | ○ | × | × | × | ENTER:G, EXITD:G |
| EaRh!M | | | | | | | JRNG:G |
| | 一般 | Rn | #imm | ESP+ | ESP- | 付加 | (対象命令) |

【図415】

8ビット符号なし 8ビット符号あり

11000000(2) = 192(10) , -64(10)

11000000(2) = 192(10) , -64(10)

11000000(2)

最上位ビット

【図395】

| | 一般 | Rn | #imm | @SP+ | @-SP | 付加 | (対象命令) |
|-------------|----|----|------|------|------|----|---|
| EaW | ○ | ○ | × | × | ○ | ○ | MOV:Z, MOV:E, MOV:C,
NOVA:C, MOVU:E, MOYU:C
PACKss, STC, STP,
UNPKss, RYBY, RYBI |
| EaWh
Shw | | | | | | | STPSB, STPSH
MOV:I, MOV:Q, MOV:S |
| EaWIS | ○ | ○ | × | × | × | ○ | MOVPA, STATE, QDEL |
| EaWmL | ○ | × | × | × | ○ | × | STM |
| EaW% | ○ | × | × | × | × | ○ | LDC:E, LDC:C,
LDP:E, LDP:C |
| EaWL | ○ | ○ | × | × | × | ○ | POP |
| | 一般 | Rn | #imm | @SP+ | @-SP | 付加 | (対象命令) |

【注】

- ① 「一般」には、@abs, @disp.PC, @disp.Rn, BRn が含まれる。
- ② レジスタ対象のビット操作命令では offset は下位ビットのみが有効
- ③ 第二オペランドのイミディエートなので変則的だが、必要性があるのでインプリメントする。

【図405】

| 命令 | 動作 |
|---|---|
| (ACB step,xreg.limit,newpc)
ACB @SP+,SP,@SP+,newpc | step: @initSP
limit: @(initSP+4)
命令実行前のxreg(=SP) initSP
EX.の直前でのxreg: initSP+8 (@SP+,@SP+ による)
命令実行後のxreg: initSP+8+@initSP
if((initSP+8+@initSP) < @(initSP+4)) jump to newpc endif |
| (SCB step,xreg.limit,newpc)
SCB @SP+,SP,@SP+,newpc | step: @initSP
limit: @(initSP+4)
命令実行前のxreg(=SP) initSP
EX.の直前でのxreg: initSP+8 (@SP+,@SP+ による)
命令実行後のxreg: initSP+8-@initSP
if((initSP+8+@initSP) ≥ @(initSP+4)) jump to newpc endif |

【図398】

| 命令 | 動作 | 実行後SP |
|-----|--|---------|
| POP | SP @initSP ==> SP
(EX で一旦 initSP+4 ==> SP となるが、
WW1 で @initSP が SP に overwrite される)
[フラグ変化を除けば MOV @SP+, SP と同じ] | @initSP |
| POP | @-SP <<モデル上は以下の動作をするが、実際は RIE>>
@ (initSP-4) ==> @ (initSP-4) initSP
(WAL で initSP-4 ==> SP, woladdr.
EX で @ (initSP-4) ==> dest2.
initSP-4+4 ==> SP.
WW1 で dest2 ==> @woladdr.
つまり @ (initSP-4) ==> @ (initSP-4))
[MOV @SP+, @-SP は @initSP ==> @initSP で動作が異なる] | |
| POP | @ (d, SP) @initSP ==> @ (d+initSP) initSP+4
(WAL で d+initSP ==> woladdr.
EX で @initSP ==> dest2.
initSP+4 ==> SP.
WW1 で dest2 ==> @woladdr.
つまり @initSP ==> @ (d+initSP))
[MOV @SP+, @ (d, SP) は @initSP ==> @ (d+4+initSP) で動作が異なる] | |

【図409】

加算命令(ADD)

| | | 32000 | 80386 | 68000 | IBM/370 | VAX | 本説明 |
|------|-----|-------|------------|-----------|---------|-----------|------------------------|
| 符号付き | 大小 | | SF(最上位ビット) | N(最上位ビット) | O | N(最上位ビット) | L(サインビット)
M(最上位ビット) |
| | 等 | | ZF | Z | O | Z | Z |
| | 符号加 | F | OF | V | O | V | V |
| | 溢出 | C | CF | C | | C | X |
| | その他 | | | X(C=無し) | | | |
| 符号なし | 大小 | | | | | | (L=0)
M(最上位ビット) |
| | 等 | | ZF | Z | O | Z | Z |
| | 符号加 | | | | | | V |
| | 溢出 | | | | | | X |
| | その他 | | | | | | |

【図 4 0 2】

| 命令 | 動作 | 実行後SP |
|--|---|----------|
| [BFINS src.offset.width.base]
BFINS SP,@SP+,SP,@(d:SP) | src: initSP+4
offset: @initSP
width: initSP+4
base: @(d+initSP+4)
initSP+4 ==> bitfield(offset,width,base) | initSP+4 |
| [BFEXT offset.width.base,dest]
BFEXT @SP+,SP,@(d,SP),RO | offset: @initSP
width: initSP+4
base: @(d+initSP+4)
bitfield(offset,width,base) ==> RO | initSP+4 |
| BFEXT @SP+,SP,@(d,SP),SP | offset: @initSP
width: initSP+4
base: @(d+initSP+4)
bitfield(offset,width,base) ==> SP
(EX に入る直前の SP 値は initSP+4 であるが、
EX で抽出されたビットフィールドが
SP に overwrite される) | bitfield |

【図 4 1 0】

乗算命令(MUL)

| | | 32 000 | 80386 | 68000 | IBM/370 | VAX | 本説明 |
|------|---------|--------------------|-------|------------------------|-----------|-------|--------------------|
| 符号付き | 大小 | | | N(符号なし) | | N | L(符号なし)
M(符号なし) |
| | 零 | | | Z | | Z | Z |
| | オーバーフロー | | OF | (V=0) | | V | V |
| | オーバーフロー | | CF | (C=0) | | (C=0) | |
| | その他 | オーバーフロー発生
にのみ対応 | | (オーバーフロー)
(オーバーフロー) | (オーバーフロー) | | |
| 符号なし | 大小 | | | N(符号なし) | | | (L=0)
M(符号なし) |
| | 零 | | | Z | | Z | Z |
| | オーバーフロー | | | (V=0) | | | V |
| | オーバーフロー | | | (C=0) | | | |
| | その他 | | | (オーバーフロー) | | | |

【図408】

| 命令 | | 命令実行後のリソース値 | |
|-------|----------------|---|--|
| STM.W | (SP), @-SP | initSP-4 ==> SP
initSP ==> @ (initSP-4)
(@-SP による)
(initSP-4 ==> @ (initSP-4) ではない) | |
| STM.W | (RO, SP), @-SP | initSP-8 ==> SP
initSP ==> @ (initSP-4)
(@-SP による)
(initSP-8 ==> @ (initSP-4) ではない)
RO ==> @ (initSP-8) | |
| STM.W | (RO, R1), @-SP | initSP-8 ==> SP
R1 ==> @ (initSP-4)
RO ==> @ (initSP-8)
(@-SP による) | |
| STM.W | (RO, SP), @SP | initSP ==> SP
RO ==> @ (initSP)
initSP ==> @ (initSP+4) | |
| LDN.W | @SP+, (SP) | initSP+4 ==> SP
(@ (initSP) の値は捨てられる)
(ポインタの更新による) | |
| LDN.W | @SP+, (RO, SP) | initSP+8 ==> SP
@ (initSP) ==> RO
(@ (initSP+4) の値は捨てられる)
(ポインタの更新による) | |
| LDN.W | @SP+, (RO, R1) | initSP+8 ==> SP
@ (initSP) ==> RO
@ (initSP+4) ==> R1
(ポインタの更新による) | |
| LDN.W | @SP, (RO, SP) | @ (initSP+4) ==> SP
@ (initSP) ==> RO
(SP の転送による) | |

【図411】

減算命令 (SUB)

| | | 32000 | 80386 | 68000 | IBM 370 | VAX | 本発明 |
|------|---------|-------|-------------|------------|---------|------------|--------------------------|
| 符号付き | 大小 | | SF (最上位ビット) | N (最上位ビット) | ○ | N (最上位ビット) | L (最上位ビット)
M (最上位ビット) |
| | 等 | | ZF | Z | ○ | Z | Z |
| | オーバーフロー | F | OF | V | ○ | V | V |
| | 桁上り | C | CF | C | | C | X |
| | その他 | | | X (CF等以外) | | | |
| 符号なし | 大小 | | | | | | L (最上位ビット)
M (最上位ビット) |
| | 等 | | ZF | Z | ○ | Z | Z |
| | オーバーフロー | | | | | | V |
| | 桁上り | | | | | | X |
| | その他 | | | | | | |

【図412】

除算命令(DIV)

| | | 32000 | 80386 | 68000 | IBM 370 | VAX | 本発明 |
|------|------|-------|-------|-----------|---------|-----------|------------------------|
| 符号付き | 大小 | | | N(最上位ビット) | | N(最上位ビット) | L(最上位ビット)
M(最上位ビット) |
| | 等 | | | Z | | Z | Z |
| | 不正符号 | | | V(0除算も) | | V(0除算も) | V(0除算も) |
| | 不正上り | | | C=0 | | C=0 | |
| | その他 | | | | | | |
| 符号なし | 大小 | | | N(最上位ビット) | | | L=0
M(最上位ビット) |
| | 等 | | | Z | | Z | Z |
| | 不正符号 | | | V(0除算も) | | | V(0除算も) |
| | 不正上り | | | C=0 | | | |
| | その他 | | | | | | |

【図413】

データ移動命令(MOV)

| | | 32000 | 80386 | 68000 | IBM 370 | VAX | 本発明 |
|------|------|-------|-------|-----------|---------|-----------|-----------|
| 符号付き | 大小 | | | N(最上位ビット) | | N(最上位ビット) | M(最上位ビット) |
| | 等 | | | Z | | Z | Z |
| | 不正符号 | | | V=0 | | V=0 | V |
| | 不正上り | | | C=0 | | | |
| | その他 | | | (不正符号-上り) | | (不正符号-上り) | |
| 符号なし | 大小 | | | | | | M(最上位ビット) |
| | 等 | | | Z | | Z | Z |
| | 不正符号 | | | V=0 | | V=0 | V |
| | 不正上り | | | C=0 | | | |
| | その他 | | | (不正符号-上り) | | (不正符号-上り) | |

【図414】

比較命令(CMP)

| | | 32000 | 80386 | 68000 | IBM/370 | VAX | 本発明 |
|------|-----|-------|-------------------|------------------|---------|------------------|------------------|
| 符号付き | 大小 | N | SF(符号)
(符号ビット) | N(符号)
(符号ビット) | O | N(符号)
(符号ビット) | L(符号)
(符号ビット) |
| | 等 | Z | ZF | Z | O | Z | Z |
| | ゼロ | | OF | V | | (V=0) | |
| | 不正 | | CF | C | | | |
| | その他 | | | | | (不正)
(符号ビット) | |
| 符号なし | 大小 | L | | | | C | L |
| | 等 | Z | ZF | Z | O | Z | Z |
| | ゼロ | | | | | (V=0) | |
| | 不正 | | | | | | |
| | その他 | | | | | (不正)
(符号ビット) | |

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☒ **FADED TEXT OR DRAWING**
- ☒ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☒ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.